Structured Intuition: A Methodology to Analyse Entity Authentication

Ahmed, Naveed

Publication date:
2012

Document Version
Publisher's PDF, also known as Version of record

Structured Intuition:

A Methodology to Analyse Entity Authentication

Naveed Ahmed

Kongens Lyngby 2012
IMM-PHD-2012-xx

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk

IMM-PHD: ISSN 0909-3192

Summary

Entity authentication is a process of verifying a claimed identity of a network party. It may appear to be a simple goal, but, depending on the application and context, it entails a number of modalities, such as whether the party is currently active on the network, whether the party is willing to communicate, and whether the party knows that it has been authenticated. Combining such goals in different ways leads to different flavours of entity authentication.

On an unauthenticated channel, an adversary can present a false claim of identity. Clearly, if the adversary succeeds, it may have serious consequences for the security of the system, e.g., private information of legitimate parties may be leaked or the security policy of a trusted system may be violated. At a corporate level, such a failure of authentication may result in loss of proprietary technology or customers' credit card information. Sometimes, a single failure of authentication affects the system for a long time, e.g., if an adversary is able to install a malicious program, such as a root kit, back door, key logger, bot, or other malware. Therefore, security protocols, which can resist a resourceful adversary, are used to authenticate network parties.

Verification of an authentication protocol to show that it is secure is a hard problem. Most of the reported flaws in authentication protocols are not due to some weakness in the cryptographic primitives used in these protocols. The usual problems lie in improper use of cryptographic primitives, and failure to unambiguously specify protocol assumptions and goals. Therefore, it is important that an authentication protocol is analysed with clear goals and explicitly stated assumptions.

There are many different formal definitions of authentication goals, and the decision of which definition is most appropriate depends on the requirements and constraints imposed by the larger system. Whether a reported flaw in a protocol is exploitable depends on the protocol goals and the environment in which the protocol is deployed. Whether a "secure" protocol is indeed secure depends on the security model and the level of abstraction used in the analysis. Thus, the goal of developing a high level methodology that can be used with different notions of security, authentication, and abstraction is worth considering.

In this thesis, we propose a new methodology, called the structured intuition (SI), which addresses the issues mentioned above. In the SI, we divide entity authentication into fine grained properties, which we call FLAGs (fine level authentication goals). FLAGs are protocol independent goals and represent one's expectations in an authentication-as-a-service paradigm. There is a single notion of security in our methodology, which is called canonicity, which is a weaker form of message authenticity. As compared to many contemporary analysis techniques, an SI based analysis provides detailed results regarding the design rationales and entity authentication goals of a protocol.

Resumé

Entitets autentificering er en process der verificerer en påstået identitet af en netværks bruger. Dette kan ses som et simpelt mål, men afhængig af anvendelsen indebærer dette en række behandlingsmetoder, som f.eks. om brugeren er aktiv på netværket, om brugeren er villig til at kommunikere, og om brugen er klar over at denne bliver autentificeret. Forskellige kombinationer af sådanne mål fører til implementationer af entitets autentificering med forskellige egenskaber.

På en ikke autentificeret kanal kan en modstander gøre et falsk krav på en identitet gældende. Hvis dette lykkes for modstanderen, kan det åbenlyst få alvorlige konsekvenser for sikkerheden af systemet, f.eks. kan private informationer vedrørende legitime brugere blive lækket eller sikkerhedspolitiken af et betroet system kan bliver krænket. På virksomheds niveau kan sådanne svigt i autentificeringen resultere i tab af proprietær teknologi eller af kunders kredit kort oplysninger. Det kan endvidere ske at et enkelt svigt i autentificeringen kan påvirke systemet i lang tid, f.eks. hvis en modstander er i stand til at installere et ondsindet program såsom et "rootkit", en bagdør, en "key logger", en "bot" eller anden "malware". På grund af dette anvendes sikkerheds protokoller til at autentificere netværks brugere, der kan modstå selv en modstander med mange resourcer.

Verifikation af en autentifikation protokol, for at vise at denne er sikker, er en svær opgave. De fleste rapporterede fejl i autentifikation protokoller stammer ikke fra en svaghed i de kryptografiske primitiver brugt i disse protokoller. Normalt skyldes problemet fejlagtig anvendelse af disse kryptografiske primitiver og manglen af en entydig specifikation af protokol antagelser og målsætning. Derfor er det vigtigt at en autentifikation protokol er analyseret med en klar

målsætning og udtrykkeligt formulerede antagelser.

Der er mange forkellige formelle definitioner af autentifikation målsætning. Hvilke af disse definitioner der er mest hensigtsmæssig afhænger af de krav og begrænsninger der er gældende i et større system. Om en rapporteret defekt i en protokol kan udnyttes afhænger i høj grad af protokollens målsætning og det miljø hvor denne protokol anvendes. Hvorvidt en "sikker" protokol faktisk er sikker afhænger af den sikkkerheds model og det niveau af abstraktion der er benyttet i analysen. Det er derfor værd at overveje udviklingen af en højniveau metode, til analyse af autentifikation protokoller, som kan anvendes med forskellige niveauer af sikkerhed, autentifikation og abstraktion.

I denne afhandling præsenterer vi en ny metode kaldet struktureret intuition (SI), der adresserer de problemer der er nævnt ovenfor. I SI inddeler vi entitets autentificering i finkornede egenskaber, som vi kalder FLAGs ("Fine Level Authentication Goals"). FLAGs definerer protokol uafhængige målsætninger der repræsenterer ens forventninger i et autentificering-som-en-service paradigme. Der er et enkelt sikkerhedsbegreb i vores metode, som vi kalder "canonicity". Dette begreb udtrykker en svagere form for besked autenticitet. I sammenligning med mange eksisterende analyse teknikker giver en SI baseret analyse et mere detaljeret billede af design rationalerne og entitets autentificering målsætningen af en protokol

(Oversat af Mads Ingerslew Ingwar)

Preface

This thesis was prepared at the Department of Informatics and Mathematical Modelling, the Technical University of Denmark, in partial fulfilment of the requirements for acquiring the Ph.D. degree in engineering sciences. The Ph.D study has been carried out under the supervision of Christian Damsgaard Jensen (DTU-Informatics) and Erik Zenner (DTU-Math) in the period of three years between June-2009 and June-2012.

The thesis introduces a new methodology for the analysis of entity authentication protocols. The methodology provides significant benefits for a system developer who is more interested in system level authentication goals and application specific security models.

The thesis consists of a summary report, eight chapters and two appendices. One of the appendices contains a collection of abstracts of the research papers written during the period 2009-2012, and elsewhere published.

Naveed Ahmed

Lyngby, June 2012

List of Published Papers

We have written the following papers in the period from 2009 to 2012. Most of these papers are partially overlapping with the contents of this thesis. We, however, recommend to use this thesis as a reference for the structured intuition methodology. The abstracts of selected papers can be found in Appendix B. One of our papers that is based on the research carried out in Fall-2011 at ETH-Zurich is not listed here, because it will be submitted in July-2012; this paper presents a new protocol for uncoordinated[1] spread spectrum communication.

1. N. Ahmed and C.D. Jensen: *Structured Intuition*, Accepted for a special issue of International Journal of Critical Computer-Based Systems, 2012
2. N. Ahmed, C.D. Jensen, and E. Zenner: *Private-key Symbolic Encryption Schemes*, In Proceedings of ESORICS-2012, Pisa, pub. Springer, LNCS Vol. 7459/2012, pp. 557–572, 2012
3. N. Ahmed and C.D. Jensen: *Post-Session Authentication*, In Proceedings of the 6th IFIP WG 11.11 International Conference on Trust Management (IFIPTM VI), Surat, India, 2012, pub. Springer, AICT Vol. 374
4. N. Ahmed and C.D. Jensen: *Security of Dependable Systems*, Book Chapter in Dependability and Computer Engineering: Concepts for Software-Intensive Systems, pub. IGI Global, 2011
5. N. Ahmed and C.D. Jensen: *Adaptable authentication model: exploring security with weaker attacker models*, In Proceedings of International Symposium on Engineering Secure Software and Systems (ESSoS), Madrid, 9-10 February, 2011, pub. Springer, LNCS Vol. 6542

[1] I.e., without relying on a shared secret between a sender and a receiver

6. N. Ahmed and C.D. Jensen: *Demarcation of Security in Authentication Protocols*, In: 1st SysSec Workshop (SysSec), Amsterdam, July 6th, 2011, pub. IEEE Computer Society, ISBN 978-0-7695-4530-1

7. N. Ahmed and C.D. Jensen: *Definition of entity authentication*, In Proceedings of 2nd International Workshop on Security and Communication Networks (IWSCN), Karlstad, 26-28 May 2010, pub. IEEE, ISBN 978-1-4244-6938-3

8. N. Ahmed and C.D. Jensen: *Entity authentication: Analysis using structured intuition*, In: 4th Nordic Workshop on Dependability and Security (NODES), Copenhagen, 3-4 June, 2010, pub. DTU

9. N. Ahmed and C.D. Jensen: *Context Aware Identity Delegation*, In Proceedings of 3rd Workshop on Combining Context with Trust, Security, and Privacy (EuroCAT09), Pisa, 2009 pub. CEUR Workshop Proceedings, vol. 504, 2009, ISSN 1613-0073

10. N. Ahmed and C.D. Jensen: *A Mechanism for Identity Delegation at Authentication Level*, In Proceedings of 14th Nordic Conference on Secure IT Systems (NordSec), Oslo, 14-16 October 2009, pub. Springer, LNCS 5838

11. N. Ahmed and C.D. Jensen: *An Authentication Framework for Nomadic Users*, In: 3rd Nordic Workshop on Dependability and Security (NODES), April 27, 2009, pub. Linköping University, Sweden,

Acknowledgements

In the last three years, I owe many people thanks, which are not all mentioned here. First, I thank Christian Jensen, for giving me the opportunity and the guidance to pursue my ideas, which helped me to accomplish things I might not have achieved otherwise; thanks for proofreading my manuscripts and helping me in improving my writing skills; it had been great working with you. I am also grateful to my other supervisor Erik Zenner, who asked me right questions, which forced me to think in precise terms. I thank my officemates, Mads Ingwar and Davide Papini, for interesting working hours in cheerful moods. I thank Mads for writing the nice Danish summary. I also spent a significant time with my good old friend Wajid Minhass, which was always a great pleasure. I thank Karin Tunder, for taking good care of everything. I also thank Ekkart Kindler, Robin Sharp, Sebastian Mödersheim, Patrick Konemann, Be-te Elsebeth, Harald Störrle, and others.

At ETH-Zurich, I met many amazing people. I specially like to thank Cas Cremers, who helped me in particular by showing interest in my work and highlighting many weaknesses. Thanks to the people at the Formal Methods group: Simon Meier, Michele Feltz, Barbara Geiser, and others. I owe thanks to Srdjan Capkun for giving me opportunity to work on an interesting project. I spent many hours working with Christina Pöpper, and learned many important things.

I very much owe my parents for supporting and helping me to reach at this stage. I thank my wife; without the ever loving support of her, things might have been different. At last, but foremost, I thank God for giving me wisdom and guidance in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Today's society depends heavily on computers. One of the benefits of these computers is their ability to communicate with each other and engage in what we call network computing. In most cases, network computing is cost-effective, due to pooling of network resources, and it provides redundancy to develop dependable services. In the future, the proliferation of network computing in every day life is only expected to increase, and our society will become ever more dependent on the correct operation of computer networks. As the famous Metcalfe's law [73, 42] states that the value of a network is a quadratic function of the number of its users, one can predict an increasing level of economic dependency on network computing.

Governments, businesses, and individuals as well as semi-autonomous computer programs use communication networks for storing, processing, and exchanging private and critical information. The basic network infrastructure normally does not provide any guarantee to a communicating party about the state of other parties on the network. Therefore, a layer of protocols is often used for reliable communication and synchronization of the states of communicating parties.

In many environments, economic and social factors may lure network users to behave dishonestly. A dishonest party—an adversary—can access and sometimes is able to control the communication. Protection against malicious activities using various network security mechanisms is among the main objectives

of building dependable systems.

An important goal of network security is entity authentication, which refers to the process where one party, the verifier, verifies the claimed identity of another communicating party, the claimant. Usually, the claimant presents its claimed identity to a verifier along with some evidence to support the claim. For instance, to login on a computer, a person normally provides his user-name, which is the claimed identity, and the password, which constitutes the evidence that the user-name belongs to the person. To protect the evidence in transit from an adversary, it is normally protected by a cryptographic scheme. In many cases, a claim of an identity is not crucial for security, and sometimes the explicit claim is not even required, such as in a non-interactive biometric based authentication.

Entity authentication is a natural requirement for communication security and many forms of access control mechanisms. Authentication not only enables a gatekeeper to prevent unauthorized parties from using a private network, but it also allows accountability on the actions of authorized parties. Without any authentication, an adversary can pretend to be an authorised party and may be able to play a man-in-middle role between honest parties. For instance, if a user does not know that whether he is connected to a legitimate net-banking portal or an adversary controlled fake website then the fake website can play a man-in-middle role between the user and the real net-banking portal. This may happen if the user does not properly authenticate his net-banking portal. As a result, the adversary may be able to steal the credentials of the user [88].

Entity authentication in a network environment is usually achieved using authentication protocols. An authentication protocol is a type of distributed program based upon cryptographic functions, which is specifically designed to allow an honest party to authenticate another honest party in the presence of an adversary.

Many factors prompt security experts to design a new protocol, for instance, a new application or a change in an application environment often implies new security requirements and trust assumptions, which may require a new protocol. A protocol also may be designed to improve the efficiency of an existing protocol or to achieve a better security guarantee. There are hundreds of published authentication protocols, e.g., the international standard ISO/IEC 9798 defines templates for seventeen authentication protocols in part 2 [146], part 3 [144], and part 4 [145]. Similarly, the survey book by Boyd and Mathuria [40] describes about two hundred authentication protocols.

In some applications, entity authentication by itself is sufficient, e.g., the domain of RFID (Radio Frequency Identification) offers many applications where only

authentication of RFID tags is required. On the other hand, most applications do require additional security goals, e.g., in secure communication both entity authentication and key establishment (for the confidentiality of subsequent data communication) are typical requirements.
In this thesis, we focus exclusively on the entity authentication goals of a protocol, even if the protocol is designed to achieve a number of other security goals.

Security analysis of communication protocols poses many intricate problems. This is especially true for authentication protocols [70], which are notoriously difficult to design and analyse. Many seemingly secure protocols have later turned out to be insecure [58, 96, 6]. Unstated assumptions and fine-level details further make entity authentication ambiguous to an end-user.

We propose a new methodology for the analysis of authentication protocols. We call our methodology the structured intuition (SI). It is a high-level methodology that analyses a protocol from the perspective of a system developer and determines which entity authentication goals are achieved by the protocol. The structured Intuition (SI) relies on some of the known techniques for low-level security analysis. It is important to note that an authentication protocol may have been designed to achieve additional security goals, such as voting or key establishment, but the analysis for those goals is currently beyond the scope of the structured intuition (SI).

A protocol analysis using intuition has a negative connotation to some people—and rightly so, because an adversary can always be more intuitive in the construction of his attack strategy. A series of sound arguments (whether formal, informal, or cryptographic) along with a well defined set of assumptions and security goals is the right way to claim security, rather than calling one's intuitive feeling about the security of a protocol. This kind of intuition must be avoided in favour of a systematic protocol analysis.

On the other hand, a great deal of intuition is required to discover non-trivial mathematical proofs, including the cryptographic proofs of entity authentication [28]. Even in an automated security analysis (such as model checking [97]), formalizing security goals, assumptions, and implementation constraints require intuition by security experts, otherwise positive results of a formal analysis maybe not correspond to actual security [134].

We use the name "structured intuition" for two reasons. First, the SI methodology consists of a number of steps, and each step has certain goals that a security analyst tries to achieve using his intuition. This means that we do not provide a mechanized (algorithmic) way to carry out each step. We rely on the intuition of the security analyst to carry out these steps and achieve the required results.

The intuition of a security analyst is used in a structured way guided by the steps of the SI methodology.

Second, during the SI based analysis of a protocol, a security analyst learns why an authentication goal that is achieved is achieved. This information helps to discover critical parts and hidden assumptions of the protocol. Further, this information makes an adaptable security analysis easier, namely starting from authentication goals a security analyst can work backwards to list the required security assumptions. The SI methodology provides a deeper insight in protocol security, and, in a sense, a complete SI based analysis represents the intuitive understanding of different aspects of a protocol.

Formulating a new methodology for the analysis of authentication protocols is in fact a meticulous undertaking with a large scope. Given the limited time frame for this PhD project, we have to make a trade-off between the scope and the depth of our research. In the scope, the SI methodology is a workable solution that enables the validation of authentication properties for a wide variety of protocols. In the depth, we describe the steps of the SI methodology in a sufficient level of detail that qualifies the presented work as *plausibly sound*.

The style of our presentation can be called light-crypto; the rigour and precisions that is expected in traditional cryptographic proofs is not used due to the large scope of our work. Similar to formal security models [62], we use symbolic abstractions for cryptographic functions, which allows us to convey the main ideas of our approach in a concise manner. We believe that it is not too hard to describe the SI methodology with appropriate computational models. In the following we present some highlights of the SI methodology.

1.1 Problem of Characterization

Entity authentication is always used as a service in a larger system, e.g., a file transfer program may rely on an authentication protocol to initialize secure transfer of data to a far-end party by verifying its identity and agreeing on a session key. Usually, a system developer is not a security expert, and therefore, he often expects the underlying security service to be a plug-and-play type, similar to other network services. Unfortunately, such an expectation from an authentication protocol can be dangerous, as we explain in the following.

Intuitively, entity authentication is a process that provides some kind of assurance to a verifier about the identity of a claimant, but when it comes to concrete requirements there seems to be no common definition of authentica-

tion among security experts [97, 40, 54, 140]. For example, the international standard ISO/IEC 9798 part 1 defines entity authentication as follows [78].

> Entity authentication mechanisms allow the verification, of an entity's claimed identity, by another entity. [And] the authenticity of the entity can be ascertained only for the instance of the authentication exchange.

Now, as per the definition, if an entity A receives a fresh signed nonce from B that A has generated earlier, should it be considered to meet the above definition?
The message is signed which satisfies the first part of the definition; and the message is fresh for A, which fulfils the second requirement of the definition. Still, this may be not satisfactory, e.g., one counter-argument is that the signed message does not convey the willingness of B to A, and, for instance, B maybe have signed the message for another party C. [1]

In fact, the term entity authentication entails different sets of requirements depending on its use in a given application. Consequently, different end-users (system developers) may interpret entity authentication differently. As Boyd states, "it is not the responsibility of the implementer to work out what a protocol achieves." [38] For security, multiple interpretations are certainly dangerous; if a system developer over-estimates the authentication goals of a protocol then any security guarantee of the protocol becomes void.

Generally speaking, a protocol goal should clearly correspond to some service-oriented authentication requirement and should not encompass superficial requirements, the so-called "accidental features." [79] Security analysis is a hard problem and trying to verify a specified goal that has little to do with actual service requirements makes little sense.

In this thesis, we rely on existing techniques of protocol analysis to get security assurance, but these techniques only solve a part of the problem. In this regard, our contribution relates to specifying what actually needs to be verified and to provide a high-level methodology that can assert a guarantee with respect to service-oriented authentication goals.

A protocol is a distributed program containing a sequence of messages exchanged between parties, and each local execution of the protocol by a protocol party is

[1] Abadi and Needham [1] also indicate this problem in the context of Denning-Sacco public key protocol [58] and address the problem in their third principle for the design of cryptographic protocols.

called a *run* of the protocol. It is natural to specify authentication requirements in terms of runs and protocol messages that are sent and received by a protocol party. In doing so, however, a number of issues arise, which make it difficult to relate high-level service requirements to low-level operational requirements on runs and messages. In the following section, we briefly look into some existing approaches to highlight the main issues that later justify our research goal. A detail comparison with the related work is presented in Chapter 8.

1.2 Historical Context

The work on authentication protocols started in 1978 with the seminal paper of Needham and Schroeder [117]. A large number of new protocols and verification techniques have been proposed since then [105, 90]. For our purpose, the work of Roscoe [133], which makes the concepts of *extensional* and *intensional* specification explicit, is most relevant here.

Intensional style captures the intent of a designer and refers to the pattern of messages and events as anticipated by the designer of a protocol. An example specification is as follows: a successful run of a protocol implies that the messages occurring in the run are all received in the same order as specified in the protocol.

On the other hand, extensional style refers to specifying protocol goals independent of the protocol. Extensional specifications are in terms of the local knowledge (or state) of a party who executes a protocol. An example specification is as follows: a successful run implies that an honest peer entity is now ready for further communication.

Most security analysts favour the intensional style [133, 28] or trace (run) based extensional style [97], because such properties can be encoded in a precise way in formal models. For example, most of the cryptographic definitions of authentication are some variation of matching conversation [59, 28, 31], which is informally defined as follows.

> At the time that Alice accepts the other party's identity (before she sends or receives a subsequent message), the other party's record of the partial or full run matches Alice's record. [59]

This intensional style definition is in terms of messages occurring in a protocol. The proof of security essentially shows that if authentication is accepted by an authenticating party then its record of the run is the same as that of the

authenticated party.
Similarly, another definition of entity authentication is as follows.

> [A property] P guarantees entity authentication of B with respect to A if whatever hostile environment is considered, it can never occur an event $commit(A, B)$ [meaning A has successfully terminated the protocol] when $run(B, A)$ [meaning B has at least started the protocol] has not occurred previously. [69]

This extensional style definition is in terms of runs of a protocol and belong to the class of trace-based properties. Its validity can be asserted, e.g., by checking all possible execution of a protocol in an adversarial environment (possibly with some boundary assumptions).

We note that none of these two definitions are service-oriented, namely it is unclear what kind of assurance is provided to the larger system if a protocol meets these definitions. While using an authentication protocol, a system developer is more interested in properties, such as whether a successful run means that the far-end party is currently present on the network and whether the far-end party knows that the local party has authenticated it. This calls for a further analysis to conclude the service-oriented goals of a protocol.

One may argue that it is easy to interpret the meaning of protocol terms to understand the service-oriented goals that a protocol achieves. We believe there are at least two problems with this argument. First, entity authentication is not a monolithic goal and in fact more than a dozen different fine-level goals correspond to two-role authentication protocols[2]. Therefore, the interpretation of authentication goals is not always straight forward. Second, if there is no formal mapping between security guarantees and service-oriented goals then it is quite possible that different system developers may arrive at different interpretations.

It is worthwhile to mention that some early forms of extensional specifications [44, 150] do have some service-oriented flavour in them. In BAN logic [44], which does not provide any definition of authentication, it is possible to come up with a specification of service requirements in terms of beliefs held by peer entities. Unfortunately, these approaches suffered from various weaknesses in their security models [39, 100]. Boyd [38] also probe this problem, but he only describes two service-oriented goals informally.

A new protocol is often introduced to address vulnerabilities identified in an existing protocol. In many cases, however, exploitation of these vulnerabilities

[2]See fine-level authentication goals (FLAGs) in Chapter 5.

may be not practical and, moreover, not all application suffer because of the identified vulnerabilities. This motivates us to find a new way to reason about the security of an authentication protocol, which incorporates the actual security requirements of an application and which considers the environment in which the protocol is intended to be deployed. For a system developer, the distinction between security and insecurity of a protocol depends on his expectations and assumptions about the protocol. In this thesis, we demonstrate that our methodology is useful in this regard, because it provides a fine grained analysis of an authentication protocol.

1.3 Fine Level Authentication Goals (FLAGs)

In this section, we outline our interpretation of entity authentication, which is from the perspective of a system developer. Our formal interpretation of entity authentication appears in Chapter 5.

Entity authentication is not a monolithic goal, instead it consists of a number of primitive goals, which we call fine level authentication goals (FLAGs). Each FLAG captures one aspect of entity authentication. In the following, we describe each FLAG from the local perspective of a verifier Alice. We use the name Bob for the party who is the claimant in the authentication process.

(1) Recognition: If Alice verifies that the claimant is the same party with whom Alice has communicated before then Alice is said to achieve the recognition of the claimant (without necessarily knowing the identity of the claimant).

Entity recognition does not require that Alice knows the claimant Bob, and therefore the real identity of the claimant is not important. An example of an entity authentication protocol is the Jane-Doe protocol [99]. We can also relate this concept to everyday life. Let us say Alice receives a handwritten document in her mailbox, and from the handwriting she may be able to convince herself that the author is the same person whose document she received last week. She does not know when these documents were written, who wrote them, and who is supposed to read them, but she can recognize the author.

(2) Identification: If Alice verifies that the claimed identity of Bob can be linked to the record of Bob in Alice's identification database then Alice is said to achieve the identification of Bob.

Continuing with the everyday scenario, let us say that Alice finds a document signed by her boss Bob, which she can identify by comparing the

signature to the known signature of Bob. She may be not able to know for whom and when Bob wrote this document (the document could be many years old). Clearly, Alice can identify Bob from the document, but she cannot establish additional modalities. In an actual protocol, this document maybe corresponds to a message that is signed by Bob, which Alice can verify using Bob's public key. An example of identification protocol is YA-TRIP [152], which is a protocol that achieves identification of an RFID tag, without achieving any other FLAG.

(3) Operativeness : If Alice verifies that the party who claims to be Bob is currently active (alive) on the network then Alice is said to achieve the operativeness of Bob.

Continuing with the everyday scenario, let us say Alice finds a report, and Alice concludes that this report was written within last ten minutes, because it contains a reference to an earthquake that occurred ten minutes ago. Now, she may not know who wrote this report and for whom this report was written, but she can establish the recentness of the report and correspondingly the operativeness of the writer.

Note that the operativeness goal does not require that Alice verifies the identity of Bob. The operativeness goal only requires that, by the end of a protocol execution, Alice has the guarantee that whoever is at the far-end is participating in the protocol execution, namely the far-end execution is not just a replay from the past. For instance, if Alice receives a message encrypted by Alice's public key and the message contains a random number that Alice has recently broadcasted then Alice can conclude that the sender is operative.

The meaning of currently active can be interpreted in slightly different manners depending on the protocol used by Alice and Bob. Sometimes different terminology is used in the literature to refer to the concept of operativeness, such as the liveness of a claimant or freshness of the evidence of authentication. The operativeness goal alone is not so useful, and it often appears in combination with other FLAGs. YA-TRAP [152], the improved version of YA-TRIP, achieves both identification and operativeness of an RFID tag.

(4) Willingness If Alice verifies that the party who claims to be Bob wants to communicate with Alice then Alice is said to achieve the willingness of the claimant.

Let us say Alice receives a questionnaire that contains specific questions about her research activities. Now, she can conclude that someone has sent this questionnaire specifically for her, although she may not know who wrote it and when it was written. The willingness goal requires that the party pretending to be Bob provides its consent to communicate with

Alice, e.g., if Alice receives a message encrypted by Alice's public key then Alice can conclude that someone has sent this message to her. Similar to the operativeness goal, the willingness goal alone is not so useful on its own, and it often appears in combination with other FLAGs, e.g., the willingness and identification goals are achieved from a signed message that contains the identity of an intendant recipient.

(5) One-sided Authentication If Alice verifies that the claimant is Bob, who is currently active and ready to communicate with Alice, then Alice achieves one-sided authentication for Bob.

Clearly, one-sided authentication occurs when the identification, operativeness, and willingness goals are achieved for the Bob. As we explain in Chapter 5, the three FLAGs (on which one-sided authentication depends) must be achieved from a single cryptographic evidence. Here the evidence corresponds to a binding sequence, which is a novel concept introduced in this thesis and defined over multiple messages of a protocol. The binding sequence is formally introduced in Chapter 4. In the literature, one-sided authentication is also referred to as single-sided or unilateral authentication.

Continuing with the everyday scenario, let us say that Alice sends a document to Bob requesting a few holidays. If after five minutes she receives the document signed by Bob then she essentially achieves one-sided authentication of Bob from the received document. This is because she had sent this request only five minutes ago, so it must be recently signed by Bob. Since the request was personalized for Alice, Bob must have known that he is signing this document for Alice, and therefore Bob's willingness can be concluded by Alice. In an actual protocol, if Alice receives a signed message from Bob that contains a recent time-stamp and the identity of Alice then Alice achieves one-sided authentication of Bob.

(6) Pseudo One-sided Authentication If Alice verifies that a recognized claimant is currently active and ready to communicate with Alice then pseudo one-sided authentication is achieved.

This goal is similar to one-sided authentication, except that the identification goal is replaced by the recognition goal. Pseudo one-sided authentication occurs when the recognition, operativeness, and willingness of a claimant are achieved.

(7) Confirmation If Alice verifies that Bob knows that a FLAG G has been achieved then Alice is said to achieve a confirmation on the goal G from Bob.

The confirmation goal is a second order FLAG, i.e., the FLAG that confirms that another FLAG is achieved. To understand its role, note that each FLAG is defined from the perspective of Alice, therefore, e.g., if the

far-end party Bob authenticates Alice then Alice cannot know about this authentication directly. For Alice, one way of determining the success of this authentication is to receive a confirmation message from Bob. This FLAG is the basis of achieving the next two FLAGs.

(8) Strong One-sided Authentication If Alice achieves one-sided authentication of Bob and further Alice receives the confirmation that Bob knows about this one-sided authentication then strong one-sided authentication is achieved.

This type of assurance is typically required in applications where a subsequent action is expected from the far-end party, e.g., without any confirmation from Bob, Alice may start streaming a TV channel to Bob while Bob is re-authenticating himself to Alice.

(9) Mutual Authentication If Alice achieves one-sided authentication of Bob, and further Alice receives the confirmation that Bob has achieved one-sided authentication of Alice then Alice is said to achieve mutual authentication for Bob.

Note that merely achieving one-sided authentication twice in opposite direction (by executing a one-sided authentication protocol twice) does not imply mutual authentication, as demonstrated by Bird et al. [30]

In our view, FLAGs represent the most common goals that a system developer might expect from the term entity authentication. We, however, do not claim the completeness of our list of FLAGs, and one may consider other goals to be relevant for entity authentication. Also note that we have not invented these FLAGs; in fact, these FLAGs appear in the literature, sometimes with different names and sometimes only implicitly. In this regard our contribution is twofold. First, we have identified these primitive goals, and we now present them in a single framework. Second, we provide a methodology that can be used to determine which FLAGs are achieved by a given protocol.

The definitions of FLAGs that we introduced in this section are useful to understand the natural meanings of FLAGs, but such informal definitions are of little value for the protocol analysis. The formal meanings of FLAGs, which we call operational definitions, are presented in Chapter 5.

It is important to make a distinction between message authentication (data origin authentication) and entity authentication. Message authentication provides an assurance that a message has not been tampered with and establish the identity of the party who have send the message. Classic techniques to achieve message authentication include message authentication codes (MAC) and digital signatures. Message authentication is an important tool that can be used

to achieve entity authentication, but similar to any other cryptographic tools (such as an encryption or hash function) it can be misused: we may not get any entity authentication even in the presence of message authentication. Message authentication is also not necessary for entity authentication. To understand this, consider the following examples.

If Alice receives a message signed by Bob's private key then Alice can conclude that this message must be sent by Bob, assuming that Bob keeps his private key secret. Now, if Alice was only interested in achieving the (timeless) identification of Bob, then this signed message is sufficient, but if her goal was to achieve the operativeness (i.e., Bob is currently there) and willingness (Bob is executing the protocol with Alice) then certainly signed message is not sufficient. For this purpose, Bob may include the current timestamp, and the name Alice in his signed message. Similarly, to achieve mutual authentication, carefully designed multiple messages are often required. On the other hand, Alice can identify Bob without relying on message authentication. For example, Bob may send a digest that is computed on the current timestamp and a secret key that is shared between Alice and Bob.

In correspondence with our fellow researchers, it seems that protocol analysts have the view that a "good" authentication protocol *must* provide the identification and operativeness (sometimes referred to as the liveness of the claimant), and the protocol should bind the claimant and verifier's identities to messages that authenticate the claimant. A "good" protocol is also expected to prevent replay and reflection attacks. This point of view is quite natural, because security experts tend to be overly cautious and they try to specify security requirements for the worst-case scenario. This extreme view representing what could go wrong is security-centric, because it does not capture the perspective of a system developer, as we explain in the following.

We believe that authentication requirements depend on the system where the authentication protocol is deployed. For instance, in many applications, reflection attacks are not possible, because a device may be single threaded and therefore only able to execute one role of the protocol[3]. In other applications, operativeness of a claimant may not be required or even possible, e.g., if a stateless device does not have a source of randomness or synchronized clock. Similarly, some applications require mutual authentication between a client and a server, and in others one-sided authentication of the client may suffice.

In our view, a protocol is good if the goals achieved by the protocol match the application requirements and constraints. A good protocol in one application is not necessarily good in another application of authentication. Therefore, a

[3]For instance, the initiator role or the responder role in a two-party protocol

better approach is to formulate a set of service-oriented entity authentication goals, which in our case are FLAGs, and then, depending on the application, a system developer can decide which FLAGs are required in his system. This means that the definition of entity authentication will depend on the system. A security analyst can determine which protocols achieve the required FLAGs. The final decision can be made by the system developer by considering the relative resource requirements and setup assumptions of the candidate protocols.

To achieve any form of entity authentication, the type and number of protocol messages depend on setup assumptions. Even with the same setup assumptions, there are many different ways to construct an authentication protocol. There is also a sheer number of different possibilities for authentication requirements. For instance, consider an authentication protocol for Alice and Bob that consists of two role programs, which are executed locally by Alice and Bob respectively. At the end of an execution, Alice and Bob can achieve different sets of FLAGs. If we only consider three FLAGs, identification, operativeness, and willingness then there are eight combinations of these FLAGs for each of the parties. This means that at the protocol level there are 63[4] different combinations for authentication requirements. Arguably, all of these sixty three combinations are useful in different applications.

1.4 Contributions

The main goal of our project is to enable a system developer to better understand entity authentication as a service and provide techniques to operationalise this service-oriented understanding in the design and verification of actual protocols. On the one hand, this will remove the vulnerabilities in the actual use of authentication, on the other hand, this will allow better trade-offs between security and resources. To achieve this goal, we address the following questions in the project.

1. What are the service-oriented goals of entity authentication?
2. Given a set of service-oriented goals, how one can verify them in actual protocols?
3. How one can achieve some kind of trade-off between goals, resources, and security?

[4]There are $8 \times 8 = 64$ combinations, but one combination represents the case when the both parties achieve no FLAG.

The contributions that are reported in this thesis are as follows.

1. We surveyed the literature to examine the goals of commonly used entity authentication protocols. As a result, we develop a hierarchy of authentication goals, which are called fine-level authentication goals (FLAGs).
2. We propose a new methodology, which we call the structured intuition (SI), which can be used to analyse a protocol for FLAGs. Further contributions are related to the SI methodology.
   (a) We introduce the concept of a dependency graph (D-graph) for modelling the functional dependencies among the messages of an authentication protocol. *A protocol as a D-graph* is not only fundamental to the structured intuition (SI) but we believe that this model is also useful in the broader field of protocol analysis.
   (b) We demonstrate that the security of an entity authentication protocol can be reduced to a simpler security goal, which we call the canonicity requirement. Roughly speaking, a message is a canonical message if it is generated by following the rules of a protocol. It is sufficient to show that certain received messages are always canonical; no other type of security analysis is required for the purpose of entity authentication.
   (c) We demarcate security and correctness requirements of an authentication protocol. A security requirement is validated by taking into account the role of a network adversary, and in the SI this corresponds to the canonicity of certain messages of a protocol. The correctness defines what is expected from a protocol by protocol users, and this corresponds to a set of FLAGs.
   (d) We illustrate that the SI methodology is useful in analysing security as an inverse problem: starting from a set of FLAGs, one tries to determine the required security assumption, such that a given protocol achieves these FLAGs. This is especially useful for analysing protocols under weaker attack models and with application specific assumptions.

One of the distinctive aspect of the SI methodology is the definition of entity authentication, which is directed to the needs of system developers. The hierarchy of FLAGs only represents authentication properties that are commonly expected from an authentication protocol, and therefore we do not make any claim about its completeness.

The SI approach is not orthogonal to cryptography or formal methods, as it relies on these methods for security analysis; the difference comes from the

actual definition of security. The SI-methodology is a high-level methodology and a security analyst can employ many existing analysis techniques within the SI framework.

1.5 Overview of Thesis

The thesis is arranged in the order in which our methodology is applied to an authentication protocol. We also use a running example alongside. In Chapter 2, we introduce the definitions of an authentication protocol and a role program. In Chapter 3, we introduce the notion of a dependency graph (D-graph), which is central in the SI methodology. In a way, a D-graph is a static model representing atomic relations among the messages of a protocol. A D-graph is constructed using a protocol narration[5]. A protocol narration is a sequence of message flows between honest parties. This is a common style of protocol specification in the literature [40].

The structured intuition consists of three main steps, as shown in Fig. 1.1.
In Chapter 4, we present step 1 and step 2. In step 1, we model a protocol from an execution perspective, namely in terms of the role programs of the protocol. A local dependency graph is a model of a role program. In step 2, we describe a way to formulate security requirements, namely which of the messages of the protocol need to be canonical. Combing the results of step 2 and step 3 results in a *binding sequence*. The notion of a binding sequence enables us to demarcate security analysis from correctness analysis.

In Chapter 5, we describe step 3 of the SI methodology. We introduce the hierarchy of FLAGs (fine-level authentication goals). The correctness analysis derives FLAGs from a binding sequence[6] of a protocol.

In Chapter 6, we apply our approach to two more protocols. In Chapter 7, we use the SI framework to solve security as an inverse problem. In particular, we apply our approach to evaluate the security of a set of RFID identification protocols. In this chapter, we also demonstrate that how trade-off between security and resources can be achieved. In Chapter 8, we compare our work to some of the existing approaches. In Chapter 9, we conclude our work with some discussion and directions for future work.

[5]The protocol narration is a common way of specifying an authentication protocol, in the form of a sequence of messages between honest parties [34, 49].

[6]A protocol may have multiple binding sequences.

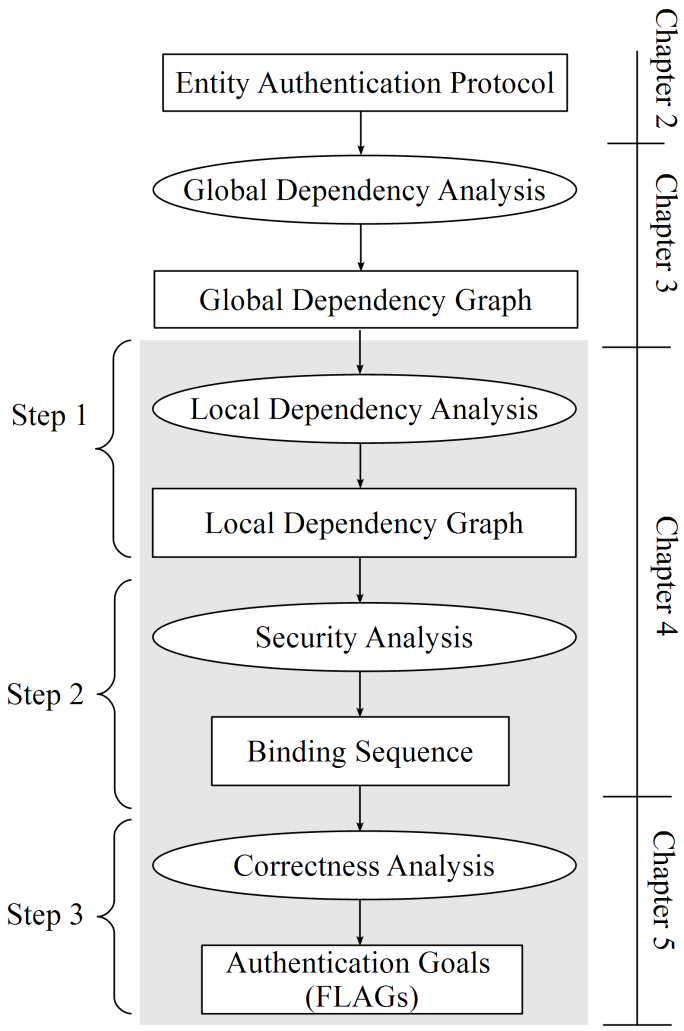


Figure 1.1: Structured Intuition

1.6 Notations and Conventions

Some of the abbreviations that we use in the thesis are listed in Table 1.1.

$\wedge$	in a predicate means *logical and*
$\vee$	in a predicate means *logical or*
:	in a predicate or set (in the set-builder notation) means *such that*
$\Rightarrow$	*material implication*
$M_i \leftarrow msg_i$	A message variable M_i on binary strings, which is assigned with a message msg_i
$M_i = M_j$	Two variables M_i and M_j contain the same value
$M_i \| M_j$	Concatenation of two binary strings that are assigned to M_i and M_j
$M_i \rightarrow M_j$	An arc from M_i to M_j in a (dependency) graph
$\mathbf{M}_i \setminus \mathbf{M}_j$	Set subtraction, i.e., $\{M : M \in \mathbf{M}_i \wedge M \notin \mathbf{M}_j\}$
$\mathbb{N}$	The set of natural numbers starting from 1: $\{1, 2, 3, \dots\}$
m	Number of role programs that constitute an m-role protocol
$\mathbb{m}$	The set $\{1, \dots, m\}$
n_0	Number of flows in a protocol narration
n_j	Number of flows in the narration of j-th role program, where $j \in \mathbb{m}$
$\mathbb{n}_0$	The set $\{1, \dots, n_0\}$
$\mathbb{n}_j$	The set $\{1, \dots, n_j\}$
$\mathcal{H}(.)$	Hash Function
$\mathcal{E}_{K_{AB}}(.)/\mathcal{D}_{K_{AB}}(.)$	Encryption/Decryption using a secret key K_{AB} shared between A and B
$\mathcal{E}_{Pk_A}(.)$ and $\mathcal{D}_{Sk_A}(.)$	Encryption and decryption using a public key (Pk_A) and a private key (Sk_A) respectively
N_A	A nonce of a party A, e.g., a sufficiently long random number, time stamp, or sequence number
R_A	A random number (R_A is a nonce and also has the property of unpredictability for network parties, including the adversary.)

Table 1.1: List of Common Notations

Chapter 2

Authentication Protocols

In this chapter, we introduce the notations, definitions, and assumptions associated with an authentication protocol, which we later use in the presentation of our methodology. An authentication protocol is specified as a narration of messages that are exchanged between the protocol parties. This is the most common style of protocol specification [34, 49]. In the following section, we start with some preliminaries, before defining a generic protocol narration.

2.1 Preliminaries

A communication network provides connectivity between parties, and enables them to send and receive messages. We use the words *party*, *entity*, and *principal* interchangeably. The letters A, B, C, S, $T_{i\in\mathbb{N}}$, and $R_{i\in\mathbb{N}}$ are variables on the identities of parties.

A *strategy* is a distributed program, which is executed by a number of network parties. We are mainly interested in two types of strategies:

1. an authentication protocol, which is a known strategy;
2. an adversarial strategy, which is an unknown strategy.

An adversarial strategy is designed by an adversary. Although we can only guess about the adversary's state of mind, we know that his strategy is constrained by his capabilities. It is typical to assume that the adversary is bounded in terms of computation power, he does not know all the secrets that are shared among network parties, and he does not have access to the sources of randomness used by honest parties to generate their random challenges. Our assumptions about the adversary are specified in the next chapter. In this chapter, we lay down the details of an authentication protocol—the known strategy.

An authentication protocol is executed by network parties while communicating with each other in the presence of an adversary. Therefore, the main challenge in the design of an authentication protocol is to provide security against any foreseeable adversarial strategy.

A party can behave in two different ways on the network. First, a party may behave honestly, namely the party follows a given protocol and respects the setup assumptions of the protocol, such as keeping its private key secret. In this case, a party can be considered as a set of trusted programs. Second, a party may not behave honestly, namely it deviates from the protocol. Such deviations may be unintentional, such as due to implementation errors or adversarial actions, but we assume that such deviations cannot occur for an honest party.

An adversary may be able to use many honest and dishonest parties to devise his attack strategy. The notion of honesty and dishonesty is only meaningful in a single session (i.e., a single execution of a protocol). A party may follow the protocol in one session, but in another session it may not follow the protocol as per an adversarial strategy.

In the structured intuition methodology, an authentication protocol is considered secure if, in a session, an honest party A of the protocol supposedly interacts with another honest party B of the protocol then A achieves a certain set of FLAGs (the protocol goals) for B. If B is dishonest then FLAGs are meaningless. This is because one cannot expect that a dishonest party will indeed be willing to communicate despite saying or even that it will keep its private-key secret.

Our notion of security can be compared to that of classic multi-party computation (MPC[1]). In MPC, it is usually assumed that the communication is on authenticated channels [12], and the main challenge of an MPC protocol is to achieve the goals of the protocol even if some of the protocol parties are dishonest. Usually, it is assumed that the majority of the parties is honest.

[1]The Chapter 7 of Goldreich's book [74] provides an excellent introduction to MPC.

In the SI, the local perspective of a protocol is important, because FLAGs are locally achieved by a party after the execution of its part of the protocol. A protocol consists of a number of role programs, which are executed by protocol parties to communicate with each other and achieve FLAGs. An example of role programs is the specification of LySa processes in LySa calculus [34]. A role program is denoted by $\rho_{j\in\mathbb{N}}(.)$. The notations A^{ρ}, B^{ρ}, S^{ρ}, $T^{\rho}_{i\in\mathbb{N}}$, and $R^{\rho}_{i\in\mathbb{N}}$ are variables on role programs.

A protocol that consists of m role programs is called an m-role protocol. In the example protocols considered in this thesis, m is either two or three, which respectively corresponds to a two-role protocol and a three-role protocol in which the third party is a trusted server.

It is quite common to use the name m-party instead of m-role, but we use the name m-role to highlight a couple of points. First, it is possible to construct a protocol in which the same role program is executed by different parties and these different parties cannot distinguish their respective roles. Thus, the number of parties executing an m-role protocol can be more than m. For example, in Boyd's conference key agreement protocol [37], there are two role programs but an execution of the protocol involves a large number of parties. Second, we draw a distinction between a *role program of a protocol* and a *party of a protocol*. This distinction is required to understand the concept of a binding sequence, which we introduce in Chapter 4.

The specification of a protocol consists of communication and functional parts. The communication part describes the sending and receiving of protocol messages between the role programs of the protocol. The functional part describes how a message that is to be sent is computed and how the verification of a received message is carried out.

2.2 Cryptographic Functions

Cryptographic functions are the main tools for designing the functional part of an authentication protocol, because they help to create asymmetry between what an adversary can do and what an honest party can do, which allows honest parties to achieve protocol goals. First we introduce the notations used for cryptographic functions and then we describe the meaning of these functions in the structured intuition.

Secret-key (or symmetric-key) encryption is denoted by $\mathcal{E}_{K_{AB}}(.)$, which stands for encrypting a plaintext of a variable length to a ciphertext, using the secret

key K_{AB} shared between A and B. Similarly, $\mathcal{D}_{K_{AB}}(.)$ denotes the corresponding decryption scheme that transforms a ciphertext back to the plaintext if K_{AB} is the correct one. Encryption and decryption using a public key Pk_A and a private key Sk_A are denoted by $\mathcal{E}_{Pk_A}(.)$ and $\mathcal{D}_{Sk_A}(.)$ respectively. The notation $\mathcal{S}_{Sk_A}(.)$ stands for the signature function using A's private key Sk_A. A hash function is denoted by $\mathcal{H}(.)$. A nonce generated by a party A is denoted by N_A. A nonce is a sufficiently long random number, time stamp, or sequence number. A random number generated by A is denoted by R_A, which is also a nonce with the additional property of unpredictability for all network parties including the adversary.

To specify the meaning of these cryptographic functions, it is important to decide the level of abstraction used for modelling these functions. We take secret-key encryption as a representative example to explain the underlying trade-off. The arguments for other cryptographic functions can be presented along similar lines.

In general, a secret-key encryption scheme enables two honest parties that share a key to privately communicate over a network, in such a way that a dishonest man-in-middle, the adversary, is unable to gain any non-trivial information about the communication. The requirements of cryptographic encryption may include left-right indistinguishability (IND) and non-malleability (NM), which can be characterized in different attack settings [91]. A common instantiation of $\mathcal{E}_{K_{AB}}(.)$ that provides both IND and NM security is the CBC[2] mode of encryption along with an message authentication code (MAC), using encrypt-then-MAC method [24].

Over the years, many abstractions of cryptographic encryption have been proposed. The most popular abstraction is the Dolev-Yao model [61]. In this symbolic model, two types of simplifications are introduced. Firstly, binary strings and functions are replaced by symbolic terms and derivation rules. In particular, this results in idealized encryption functions—either an adversary can decrypt a symbolic ciphertext (e.g., if he can derive the key) or the adversary gets absolutely no information about the plaintext. The second simplification is related to the capabilities of an adversary, namely the adversary is modelled as a non-deterministic strategy that is limited to selecting its actions from a small set of (pre-defined) logic rules. The security models that use these two abstractions are commonly referred to as symbolic/formal security models.

A symbolic model is simpler than its cryptographic counterpart, and therefore one can avoid relatively complicated and long proofs of traditional cryptography. More importantly, computers can do the tedious job of proving (and similarly verifying) the proofs of security. Unfortunately, any security assurance in a sym-

[2]Cipher-block chaining [64]

bolic model does not automatically translate to the underlying computational cryptography and, therefore, to its hardware/software implementation. In any implementation of symbolic encryption, a system designer has to make certain security critical decisions, related to, e.g., mode of encryption, block alignment, and message authentication code. Many attacks targeting the implementation of encryption are known [29, 125]. Not long ago, many research efforts spurred to address this obvious gap between symbolic and computational cryptography, most notably, Abadi and Rogaway [3], and Backes, et al. [10] independently published interesting initial results.

The SI methodology is not tied to a particular level of abstraction, such as Dolev-Yao model [61] or Bellare-Rogaway model [28]. Also, different steps of the SI can be carried out with different level of abstractions. In the presentation of the SI in this thesis, we use symbolic abstractions of cryptographic functions, so that our main ideas can be conveyed in their simplest forms. This means that the default assumptions are as follows, which we make stronger by considering only a subclass of all polynomial-time adversaries:

- We assume the model of a cipher for both secret-key and public-key encryption schemes, namely the mapping between a plaintext and its ciphertext is defined by a pseudo-random permutation (PRP).

  Note that the requirements for a cipher are different from that of an encryption scheme, because a cipher or PRP is not a randomized function, and therefore it does not provide IND-security (or semantic security) [91]. A cipher generates the same ciphertext if input is the same, but a cipher generates a random ciphertext for a new plaintext. An encryption scheme, such as CBC or CFB, is based on a cipher and uses an initialisation vector, which randomises the output of the encryption scheme.

  The output of a cipher is non-malleable however. For example, this means that $\mathcal{E}_{K_{AB}}(N_A \| N_B)$ cannot be modified to generate $\mathcal{E}_{K_{AB}}(N_A \| N'_B)$, where $N_B \neq N'_B$, without the knowledge of K_{AB}.

  Although the assumption of *a PRP as a cipher* is used in traditional cryptography, there is an important difference between our assumption and the cryptographic one. In cryptography, a cipher is of fixed length, and this fixed length is called the block size of the cipher, e.g., for AES the block size is 128 bit. On the other hand, in our assumption, a plaintext can be of any length, because we are using a symbolic abstraction, e.g., we say there is a nonce N_A but we never specify the length of N_A. The length of a plaintext can span the multiple blocks. Therefore, our assumption is much stronger than that of traditional cryptography.

- We assume the random oracle model [25] for a hash function $\mathcal{H}(.)$, i.e., a digest (the output of a hash function) has the uniform distribution.

- Similarly, we assume the random oracle model for a signature function $\mathcal{S}_{Sk_A}(.)$, i.e., the signature of a message has the uniform distribution.

With these assumptions, there is always a negligible probability of errors associated with cryptographic functions, e.g., on seeing a ciphertext an adversary, who does not know the key, can guess the plaintext with at least a negligible probability (in the security parameter of the encryption function).

On the other hand, we make our presentation of the SI even simpler by introducing more abstraction: we define an adversary model for which we can avoid writing the negligible probability of errors associated with PRP and random oracle models. The adversary model that is assumed in the SI is defined in the next chapter.

Next, we define the notion of a message variables, which is extensively used in an SI based analysis.

Definition 2.1 (Message Variable) A variable M of a protocol is called a message variable if it has the following properties:

1. (Semantics) The variable M is on the range of a known probabilistic function.
2. (Communication) The value of M is sent or received on the network as per the protocol specification.
3. (Length) The length of the message variable is fixed and a priori known.

The value of a message variable is called a message, which is communicated over the network during an execution of the protocol. Clearly, a message is an atomic value in a sense that it cannot be further parsed into meaningful components. A few examples of message variables are as follows:

- The value of a variable A, which is on the identities of network parties, may represent a received message in a protocol. In such a case, A is a message variable, whose semantics are determined by the uniform distribution on the identities of network parties. Also note that the value of A is an atomic value.

- A protocol may have a message variable $M = \mathcal{H}(A\|N_A)$. The value of $\mathcal{H}(A\|N_A)$ is an atomic value, where $\mathcal{H}(.)$ is a hash function and N_A is a nonce. The value of $\mathcal{H}(A\|N_A)$ cannot be further parsed into two meaningful values. The semantics of M are defined by $\mathcal{H}(A\|N_A)$.

- A protocol may have a message variable $M = \mathcal{E}_{Pk_A}(A\|N_A)$. The value of $\mathcal{E}_{Pk_A}(A\|N_A)$ is an atomic value, where $\mathcal{E}_{Pk_A}(.)$ is a public-key encryption function. The value of $\mathcal{E}_{Pk_A}(A\|N_A)$ cannot be further parsed into two meaningful values. Note that the values of A and N_A cannot be obtained by parsing the binary string corresponding to $\mathcal{E}_{Pk_A}(A\|N_A)$.
- A variable $M = \mathcal{E}_{Pk_A}(A\|N_A), \mathcal{E}_{Pk_A}(N_A)$ cannot be a message variable, because the value of $\mathcal{E}_{Pk_A}(A\|N_A), \mathcal{E}_{Pk_A}(N_A)$ consists of two atomic values.

2.3 Protocol Narration

A common way of specifying an authentication protocol is in the form of a protocol narration [34, 49]. A protocol narration consists of a sequence of flows:

Definition 2.2 (Flow) A flow[3] represents a set of messages that can be sent from a sender to a receiver in parallel.

Now, we define a generic protocol narration, which consists of n_0 flows.

Definition 2.3 (Generic Protocol Narrations Π) For $1 \leq i \leq n_0$, let

1. the list of m role programs be $\mathbf{Roles} = \rho_1(.), \ldots, \rho_m(.)$.
2. $\mathbf{M}_i$ be a list of all message variables in the i-th flow.
3. $F_i \stackrel{\text{def}}{=} T_i^\rho \longrightarrow R_i^\rho : \mathbf{M}_i$, i.e., in the i-th flow $\mathbf{M}_i$ is sent by $T_i^\rho \in \mathbf{Roles}$ and received by $R_i^\rho \in \mathbf{Roles}$.

A generic protocol narration is as follows:

$$\Pi \stackrel{\text{def}}{=} [F_i : 1 \leq i \leq n_0]$$

In simple words, a sequence of flows between m role programs is called a protocol narration of an m-role protocol. A protocol narration is denoted by Π. A protocol narration mainly describes the communication part of the protocol specification. Usually, a protocol narration also provides enough information about the functional part so that a security expert can write detailed versions of the role programs of the protocol.

The list of message variables in a protocol narration is denoted by $mv(\Pi)$, i.e., $mv(\Pi) = \mathbf{M}_1, \ldots, \mathbf{M}_{n_0}$; note that the message variables are place holders

[3]The term *flow* is also used in the same context in Bellare-Rogaway model [28].

for messages, i.e., the values that are sent or received are assigned to message variables. If a message has multiple receivers then multiple flows can be used to specify such a transmission.

In this thesis, we consider two-role protocols ($m = 2$), and three-role protocols ($m = 3$) that include the role of a trusted third party. Our theoretical model assumes an arbitrary value of m, but the validation with $m > 3$ is left as a part of future work.

2.3.1 Case Study 1: Protocol Narration

As a running example, we consider the five-pass authentication protocol from the international standard ISO-9798 part 2 [146, 51]. The protocol is a three-role protocol based on symmetric-keys and the notion of a trusted third party. The protocol consists of five flows among three role programs. The protocol narration, without optional text fields, is as follows:

(1) $A^\rho \longrightarrow B^\rho$: $M_1 = R_A$
(2) $B^\rho \longrightarrow S^\rho$: $M_2 = R'_B, M_3 = R_A, A$
(3) $S^\rho \longrightarrow B^\rho$: $M_4 = \mathcal{E}_{K_{BS}}(R'_B \| K_{AB} \| A), M_5 = \mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B)$
(4) $B^\rho \longrightarrow A^\rho$: $M_6 = \mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B), M_7 = \mathcal{E}_{K_{AB}}(R_B \| R_A)$
(5) $A^\rho \longrightarrow B^\rho$: $M_8 = \mathcal{E}_{K_{AB}}(R_A \| R_B)$

The protocol narration consists of eight message variables, five flows, and three role programs. The initiator role program is A^ρ, the responder role program is B^ρ, and a server role program is S^ρ. A party A plays the initiator role and executes A^ρ. In flow (1), the program A^ρ sends a random number R_A (as a nonce) to another role program B^ρ. The party B plays the responder role and executes the role program B^ρ. In flow (2), the program B^ρ forwards the received nonce along with its own nonce R'_B to another role program S^ρ. A trusted party S executes the program S^ρ.
The trusted party S is assumed to share long term keys with each of the network parties, including A and B. With the party A, the shared key is K_{AS}, and with the party B, the shared key is K_{BS}. In flow (3), the role program S^ρ sends two ciphertexts to B^ρ, one for B and one for A, containing a session key K_{AB} and respective nonces. In flow (4), the program B^ρ forwards A's ciphertext along with a new ciphertext created using K_{AB} and containing a new random number R_B. In the last flow, the program A^ρ sends back R_B and R_A encrypted using the session key K_{AB}.

2.4 Role Programs

An m-role protocol is a distributed program that is to be executed by a number of network parties while communicating in the presence of an adversary. The SI methodology relies on the local perspective of a protocol party to draw conclusions about the overall state of the protocol execution. A role program is the part of the computation and communication carried out by one of the protocol parties.

It is also possible to write a "universal" role program that emulates any of the m role programs at run-time. To exclude this possibility, we assume that a role program always sends and receives the same types of messages in each execution. Note that a dishonest party does not execute a role program; the malicious program of the dishonest party, however, could be a slight variation of a role program, such as swapping of two messages in a flow.

Similar to a protocol narration, we specify the communication part of a role program as a role narration.

Definition 2.4 (Role Narration) A role narration $\Pi_{j\in\mathbb{m}}$ of the j-th role program $\rho_j(.)$ is a subset of the protocol narration, $\Pi_j \subseteq \Pi$, such that, for each flow $F_i \in \Pi_j$, either the sender of F_i or the receiver of F_i is $\rho_j(.)$.

A role program does not necessarily participate in each flow of the protocol. The flows in which a role program does not participate as a sender or receiver are not included in the role narration.

The execution model that we envisage for a party is shown in Fig. 2.1, which is presented as a sequence diagram. In the figure, three processes are shown: a calling routine, a role program $\rho_j(\overline{\mathbf{const}}_j)$, and a communication interface. The calling routine provides an interface to a role program for higher level applications, and the communication interface represents a transport layer that ensures reliable inter-party communication over a network.

To start an execution, either the calling routine receives a request from a peer entity to execute a role program, as indicated by the arrow with label (1), or the calling routine starts a run by itself after receiving a request from an application. For example, the former case occurs if the peer entity is running the initiator program in a two-role protocol, and the latter case occurs if a user requests to authenticate a peer entity.

The calling routine invokes a copy of a role program, as illustrated by the arrow

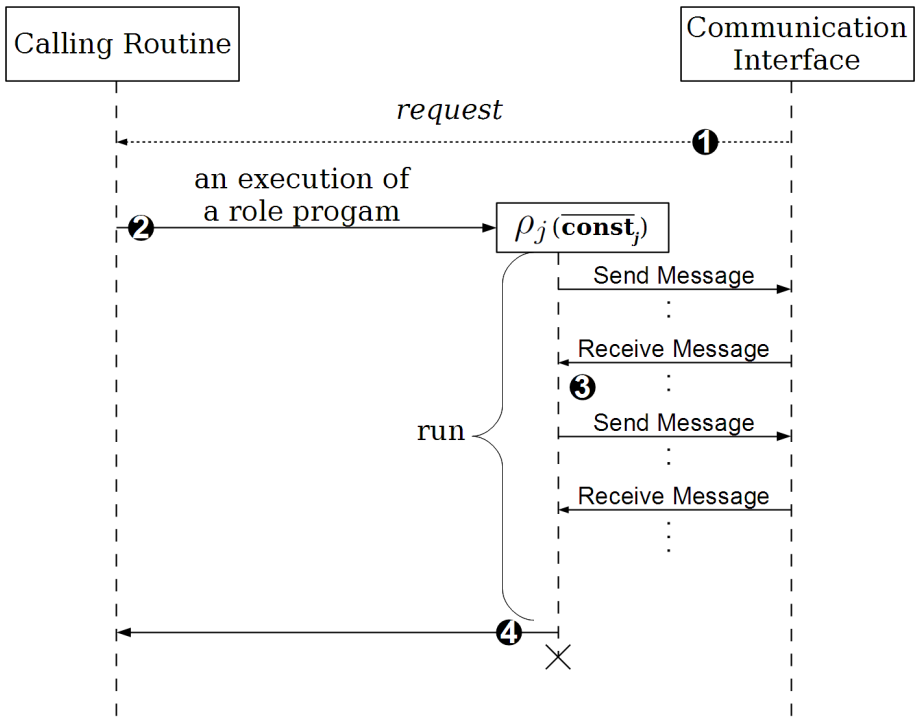


Figure 2.1: Execution model: r^{th} run of a role program

with label (2) in Fig. 2.1. A single execution of a role program is called a *run*. A role program $\rho_{j\in\mathrm{m}}(\overline{\mathbf{const}}_j)$ is invoked with a list of constants $\overline{\mathbf{const}}_j$, which may include the values of long term keys, public keys of peer entities, assumed identities of peer entities, and the random numbers that are to be used in the run[4]. A constant is a known value at the start of a run. We use a *bar* on a term that is a constant in a role program, e.g., $\bar{A}$ for an identity that is supplied as a constant to a role program. A role program may have a number of internal variables, but in the SI only message variables are important.

A party may execute multiple copies of the same or different role programs in parallel. A role program sends and receives messages using the communication interface, which provides low level communication services, such as transmission control and routing. This communication phase is indicated by label (3) in Fig. 2.1.

[4]This concept of providing input to a role program is similar to the initial knowledge in an AnB specification [112].

The list of message variables of a role program $\rho_j(.)$, whose role narration is Π_j, is denoted by $mv(\rho_j(.))$ or $mv(\Pi_j)$.[5] If Π is the corresponding protocol narration then

$$mv(\rho_j(.)) = [x : x \in mv(\Pi), receiver(x) = \rho_j(.) \vee sender(x) = \rho_j(.)]$$

We distinguish between the variables corresponding to received messages and sent messages. The list of variables for received messages is denoted by $\mathbf{Rx}_j$, and that for sent (or transmitted) messages is denoted by $\mathbf{Tx}_j$. In $\rho_j(\overline{\mathbf{const}}_j)$, the values that can be computed from $\overline{\mathbf{const}}_j$ and $\mathbf{Rx}_j$ are called *known values*. A variable in $\mathbf{Tx}_j$ is always assigned with a known value.

We require that a message variable in $\mathbf{Rx}_j$ is assigned with a semantically correct, received message (See Def. 2.1). The semantic correctness[6] of a message is ensured by following the functional requirements as specified in the protocol narration. For instance, if $\rho_j(.)$ expects to receive a message for a ciphertext $\mathcal{E}_K(N_A \| N_A + 1)$ and K is known then any received message that is decrypted to $msg_1, msg_1 + 1$, where $|msg_1| = |N_A|$, is a semantically correct message. If, however, the key K is not known then any message with the right length, i.e, $|\mathcal{E}_K(N_A \| N_A + 1)|$, is a semantically correct message. If such a semantic check fails then the program does not terminate immediately. We assume a time-out mechanism that waits for a semantically correct message for a pre-specified period of time. Although semantic checks are not important in our methodology, they are necessary to avoid trivial denial of service attacks, e.g., to protect against random messages that can be easily sent by an adversary.

We assume independent runs, namely no two runs communicate with each other locally[7] or share common variables. We also assume that all terms of a protocol are distinct from the terms of any other protocol that is executed in the network, namely we do not consider secure composition of different protocols. A role program completes its communication part if one of the following events occurs:

- A time-out event occurs that signals that the waiting time for the arrival of a semantically correct message is elapsed.
- All message variables in $\mathbf{Rx}_j$ have been assigned.

In the former case, the run terminates with an error message. In the latter case, the run verifies certain dependency relations among the messages, which we describe in Chapter 4. If the verification fails, the run must terminate with

[5] With the slight abuse of notations, we are using the same functional notation $mv(.)$ to denote message variables of a role program and that of the protocol, but this is essentially harmless, as the distinction between the two cases is always clear from the context.

[6] The requirement of semantic correctness is similar to pattern matching [34].

[7] Note that this does not exclude the possibility of a reflection attack, in which two runs on the same party communicate via an external network.

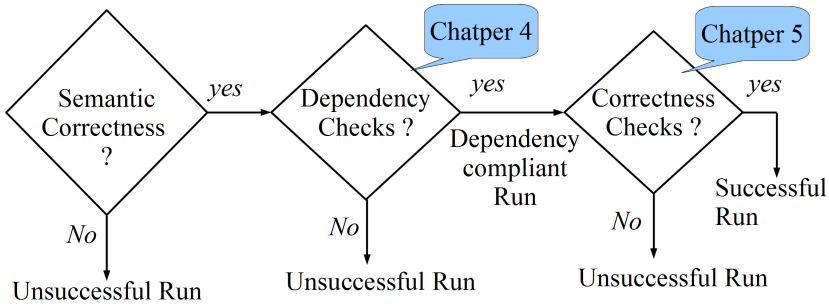


Figure 2.2: Successful run in a flow chart

an error message. If the verification succeeds then certain correctness checks are carried out; the correctness checks are described in Chapter 5. If the correctness checks also succeed then the run is called a successful run. This flow is also illustrated in Fig. 2.2.

A successful run generates a list of outputs $\mathbf{Output}_j$, as indicated by the arrow with label (4). The contents of $\mathbf{Output}_j$ depend on the authentication goals of the role program, e.g., $\mathbf{Output}_j$ may consist of the identity of the peer entity that participated in the run. Usually, the role programs that only achieve entity authentication (without achieving additional goals such as key establishment) generate a binary output corresponding to the success or failure of a run.

We assume an honest party uses its authentication credentials (e.g., private keys) only for an authentication protocol. We assume that the implementation of an honest party is secure[8], and that an adversary cannot directly access the internal states of an honest party. Similarly, the security issues due to side channels are considered outside the scope of our model, e.g., timing, power, electromagnetic and acoustic analysis[9]. We also do not consider an adversary who is interested in denial-of-service (DoS) attacks.

2.4.1 Case Study 1: Role Programs

For our example protocol (ISO-9798 part 2 [146, 51]), the narration of a role program can easily be obtained from the protocol narration (Page 26), by omit-

[8]For small systems, one may be able to remove this assumption by including the analysis for secure information flow, e.g., using type systems [155], to ensure the safe behaviour of all programs.

[9]`http://tau.ac.il/~tromer/acoustic`

ting those flows in which the role program is not a sender or a receiver. In this way, the narration of the initiator program A^ρ is as follows.

$$
\begin{array}{lll}
(1) & A^\rho \to B^\rho: & M_1 = R_A \\
(4) & B^\rho \to A^\rho: & M_6 = \mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B), M_7 = \mathcal{E}_{K_{AB}}(R_B \| R_A) \\
(5) & A^\rho \to B^\rho: & M_8 = \mathcal{E}_{K_{AB}}(R_A \| R_B)
\end{array}
$$

The narration of the responder program B^ρ is the same as the protocol narration, because B^ρ takes part in all the flows. The narration of the server program S^ρ is as follows.

$$
\begin{array}{lll}
(2) & B^\rho \to S^\rho: & M_2 = R'_B, M_3 = R_A, M'_3 = A \\
(3) & S^\rho \to B^\rho: & M_4 = \mathcal{E}_{K_{BS}}(R'_B \| K_{AB} \| A), M_5 = \mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B)
\end{array}
$$

Chapter 3

Dependency Graph

In this chapter, we introduce the notion of a dependency graph (D-graph), which is fundamental to the structured intuition (SI) methodology. A D-graph is a type of digraph (directed graph). We say a digraph is weakly connected if the underlying undirected graph is connected[1]. A D-graph is a weakly connected digraph that is constructed using a set of protocol messages.

The structure of a D-graph represents certain functional dependencies among protocol messages. In this chapter, we present D-graphs using a global perspective, namely we assume that one can freely use all of the messages of a protocol in the construction of a D-graph. This means that we do not associate the resultant D-graph to a particular role program of the protocol.

The global perspective, however, is not an operational view of protocol execution. An association between a D-graph and a protocol party is essential so that the party can verify the D-graph, but this concern is not relevant in the global perspective that we use in this chapter.

The main purpose of using the global perspective is to introduce the reader to different components of a D-graph and provide an intuitive feeling of its relevance to entity authentication. A local perspective of protocol execution

[1] A strongly connected digraph is a digraph in which it is possible to reach any node starting from any other node by traversing arcs.

is presented in the next chapter. The distinction between the global and local perspectives is the same as pointed out by Steve Schneider [138] between a user's view and a "God's view."

3.1 Dependency Function

Let us consider two variables M_1 and M_2, such that M_2 is a function of M_1: $M_2 = f(M_1)$. A dependency function (D-function) captures the intuition that if a party can compute the value of M_2 by computing $f(.)$ then the party must have the complete knowledge of M_1. In other words, M_2 *depends* on M_1, and there is no feasible way to compute M_2 without knowing the value of M_1.
For example, $f(.)$ can be the square root function, and clearly a party cannot compute the square root of a number without knowing the number. This dependency condition does not hold for every function, e.g., $f(.)$ can be defined as the least significant bit (LSB) of the input, and in this case one can compute M_2 (the LSB of M_1) without completely knowing M_1.[2] A cryptographic hash function is a D-function, which also serves as a good mental model of D-functions.

In this section, we generalize the concept of dependency and introduce a dependency assumption and a generic D-function.

Definition 3.1 (Dependency Assumption) Let a collision in a function $f(.)$ be an event of finding *input* and *input*$'$, such that $f(input) = f(input')$ for $input \neq input'$. An adversarial strategy that finds a collision in $f(.)$ does not succeed.

The dependency assumption holds for most of the functions used in the construction of authentication protocols, e.g., encryption, decryption, and identity functions are injective functions. For non-injective cryptographic functions, such as secure hash and signature functions, the probability of a collision is conjectured to be small, and therefore the dependency assumption holds for these functions with an overwhelming probability.

A commonly used function in authentication protocols is the exclusive-or (Xor) function, however, the Xor function of two arbitrary messages does not preserve the dependency assumption. If $M_3 = M_1 \oplus M_2$ then there are a large number of values for M_1 and M_2 that result in the same output, e.g., for two bit vectors $00 \oplus 11 = 01 \oplus 10 = 10 \oplus 01 = 11 \oplus 00$. On the other hand, if either M_1 or M_2

[2] In this example, an adversary may be able to replace the original value of M_1 with the value of his choice M_1' such that M_1 and M_1' have the same LSBs.

is a known constant in a protocol then the Xor function reduces to an injective function, for which the dependency assumption holds trivially.
Next, we define a dependency function using the dependency assumption.

Definition 3.2 (Dependency Function) A function $y = Dep(x, \mathbf{aux})$ is called a dependency function (or D-function) if

1. $y = Dep(x, \mathbf{aux})$ is a non-deterministic polynomial time function, where x is a variable on binary strings, and **aux** is a possibly empty list of variables on binary strings.
2. the dependency assumption of $Dep(.)$ holds with a probability of $1 - \epsilon$, where ϵ is a negligible function in the size of the security parameter of the D-function.

We say that y *depends on* x if $y = Dep(x, .)$, and, since an adversary may have non-zero probability of finding a collision in $Dep(.)$, the dependency of y on x is interpreted in a probabilistic sense. For instance, if $y = \mathcal{H}(x)$ then the dependency of y on x cannot hold with a probability of 1, due to the possibility of finding collisions in the hash function.

In Def. 3.2, x and **aux** are respectively referred to as the *main argument* and the *auxiliary argument*. The reason to distinguish between the domains of x and **aux** for a D-function will become clear in the next section where we use D-functions to construct dependency graphs. Roughly speaking, x represents a message variable that is assigned with a message that is communicated between two protocol parties. On the other hand, **aux** stands for any data that is necessary to compute the D-function.
A few examples of a D-function are as follows:

1. If $M_1 = \mathcal{E}_{Pk_A}(R_A)$ and $M_2 = \mathcal{E}_{Pk_A}(R_A{+}1)$ then $M_2 = Dep_1(M_1, Sk_A, Pk_A)$ and $M_1 = Dep_2(M_2, Sk_A, Pk_A)$ are the D-functions from M_1 to M_2 and from M_2 to M_1 respectively, where Pk_A and Sk_A are a public/private key pair:

$Dep_1(M_1, Sk_A, Pk_A)$:	$Dep_2(M_2, Sk_A, Pk_A)$:
(1) $x_1 \leftarrow \mathcal{D}_{Sk_A}(M_1)$	(1) $x_1 \leftarrow \mathcal{D}_{Sk_A}(M_2)$
(2) $x_2 \leftarrow \mathcal{E}_{Pk_A}(x_1 + 1)$	(2) $x_2 \leftarrow \mathcal{E}_{Pk_A}(x_1 - 1)$
(3) return x_2	(3) return x_2

   Note that for a D-function, there is no restriction regarding who should be able to compute the D-function. In this example, the D-functions can only be computed by A, because only A knows his private key Sk_A, but this is not relevant to the definition of a D-function.

2. If $M_1 = \mathcal{E}_{Pk_A}(R_A \| R_B)$ and $M_2 = \mathcal{E}_{Pk_A}(R_A + 1)$ then no D-function exists from M_1 to M_2, because M_2 does not depend on R_B, and therefore collisions can be generated by changing the value of R_B. On the other hand, there is a D-function from M_2 to M_1:

   $Dep(M_2, R_B, Pk_A, Sk_A)$:
   (1) $x_1 \leftarrow \mathcal{D}_{Sk_A}(M_2)$
   (2) $x_2 \leftarrow \mathcal{E}_{Pk_A}(x_1 - 1 \| R_B)$
   (3) return x_2

3. If $M_1 = R_A$ and $M_2 = R_A + 1$ then $M_2 = Dep(M_1)$ and $M_1 = Dep(M_2)$ are the D-functions, which are defined by the increment function and the decrement function respectively.

A mental trick to locate a D-function is as follows: only if each part of M_1 is necessary to compute M_2 then the function from M_1 to M_2 may be a D-function of the form $M_2 = Dep(M_1, .)$. The main requirement for $M_2 = Dep(M_1, .)$ is the same, i.e., the dependency assumptions of $M_2 = Dep(M_1, .)$, as per Def. 3.1, must hold. In the first example above, all parts of M_1 must be known to compute M_2, and vice versa. In the second example, R_B is not required to compute M_2.

3.1.1 Behind the Abstraction of a D-function

There are a few aspects of the above definition that deserve further elaboration, namely the generic nature of the definition, non-determinism, and the role of cryptographic keys in D-functions. First, in Def. 3.2, it is not specified how x and **aux** are assigned in a protocol. In fact, the way x and **aux** are assigned makes the above definition generic:

- An injective function is trivially a dependency function, since there cannot be any collisions in the injective function.
- For a non-injective D-function, the dependency assumption may hold with a high probability.
  - If x and **aux** can be freely assigned by an adversary then the dependency function needs to be "collision resistant" in a traditional cryptographic sense [132], e.g., this case occurs if a party receives the digest of a commitment and the commitment is to be revealed later in the protocol.
  - If x and **aux** are assigned by an honest party then the dependency function needs to be a second preimage resistant function [132], e.g., if a party receives a signature on the nonce that the party has sent earlier.

It is known that collision resistance implies second preimage resistance, but collision resistance does not guarantee onewayness (preimage resistance) of the function [132]. This means that a non-injective D-function is not necessarily a oneway function.

It is important to note that the dependency assumption holds due to the nature of a function and the type of function arguments. Therefore, some functions of a protocol are inherently D-functions and others are not, and it does not matter whether a particular party (e.g., an adversary) can or cannot compute them.

The second aspect of Def. 3.2 may seem unrealistic, because it allows non-deterministic functions. Clearly, non-determinism implies that a dependency function may not be even computable by a protocol party, because parties are not expected to compute any super-polynomial function.

A non-deterministic function, however, can be verified in a polynomial time if the party knows the correct values of non-deterministic choices required to compute a D-function. These choices are usually referred to as an NP-certificate. In fact, an efficient verification of a D-function is precisely the requirement in our methodology. This requirement is stated in the next chapter when we discuss the execution of role programs.

As an example of non-determinism, consider a party A that receives two messages, $M_1 = N_A$ and $M_2 = \mathcal{E}_{Pk_A}(N_A \| N_B)$. If the nonce N_A is known to A, but the nonce N_B is not known, then A cannot compute M_2 by only using M_1 in the encryption function. The party A, however, can compute M_1 from M_2 using the corresponding decryption function, but the function that maps M_2 to M_1 using the decryption function is not a D-function in this particular case. This is because one can generate a large number of values for M_2 that map to M_1 by simply changing the value of N_B.

In this example, the encryption function is a dependency function, because one cannot generate two values of M_1 that maps to the same value of M_2. The dependency function based on the encryption function is clearly non-deterministic due to unknown value of N_B; N_B is the NP-certificate of the dependency function. Nevertheless, the party can easily verify the dependency function using the decryption function.

The verifiability of a D-function is discussed in the next chapter, where we use the dependency functions to model the role programs of a protocol. Since we are using a global perspective in this chapter, the issue of non-determinism is not relevant—an NP-certificate can always be a part of the auxiliary arguments of a D-function.

The third aspect that is worth considering is the role of a cryptographic key in a D-function. For example, consider an encryption function, which can be written in two different ways: $\mathcal{E}_{K_{AB}}(N_A)$ or $\mathcal{E}(K_{AB}, N_A)$. The first style indicates a family of functions from which a specific function is selected using the secret key K_{AB}. The second style indicates that there is just one encryption function whose first argument is the secret key. In the first style, the encryption function is written as an injective function, while in the second style, the encryption is written in a form that represents a non-injective function.

Indeed both styles are correct ways of specifying cryptographic encryption, however, they lead to different forms of D-functions and D-graphs. For the first style, a key is always implicit and never appears as a main or auxiliary argument of a D-function, and for the second style, the key is treated as an ordinary function argument. A long term key, which is a priori known, can be made implicit in a D-function, because such a key is always a part of the local knowledge of a party and can be used for later derivation of authentication goals. Both styles can be used in the modelling of a protocol in our SI methodology, and both leads to the same authentication properties, although with slightly different formalizations of intermediate steps.

In this thesis, we prefer the first style, in which a long term cryptographic key appears implicitly and is not treated as an ordinary function argument. This choice not only results in simpler versions of D-graphs but this style is also intuitively appealing[3]. The reader must note that keys that are freshly generated during a session must be treated as ordinary arguments, e.g., in $\mathcal{E}_{K_{AB}}(N_A \| K'_{AB})$ the key K'_{AB} is an ordinary argument similar to N_A.

When using a key as the main argument of a D-function, one must be careful however. For example, consider two messages $M_1 = K_{AB}$ and $M_2 = \mathcal{E}_{K_{AB}}(R)$. One may be tempted to treat this encryption function as a D-function with R as an auxiliary argument: $M_2 = Dep(M_1, R)$. Unfortunately, the resultant function is not a D-function. This is because any value of M_1 can be mapped to a fixed value of M_2 by using the right value for R. Also note that $Dep(M_1, R)$ is a non-injective function.

In a network environment, there are two ways in which an adversary can compute the output of a dependency function. Firstly, he can compute a dependency function locally if he knows the correct values of its arguments, which may include private keys of honest parties. Alternatively, an adversary can rely on the

[3] A cipher is traditionally modelled as a family of functions. A value of the key represents the choice that a party makes and this value is then treated as a constant within a protocol execution. The subscript notation in the first style represents this fact that the key is fixed. Traditionally, the arguments of a function are variables, and therefore it is natural to not treat the key as a function argument.

network, namely he may able to use other parties to compute a D-function, e.g., by eavesdropping on the communication between honest parties or by using an honest party as an oracle.

3.1.2 Adversary

The SI methodology can be applied with different (application specific) adversarial models, as long as the dependency assumption holds, i.e., an adversary cannot feasibly find collisions in a D-function. For entity authentication, a passive adversary, who only eavesdrop on a network, is essentially harmless. Therefore, a realistic adversary is more powerful than a passive adversary. There is an upper limit however. The upper limit corresponds to the most powerful attacker that can be safely used in the presented version of the SI methodology. If the capabilities of an adversary do not respect this upper limit then some of the results presented in the thesis may not hold. In the following, the adversary is defined as a class of strategies $\mathbf{I}_0$.

Definition 3.3 (Adversary Class $\mathbf{I}_0$) Let the class $\mathbf{I}$ be the set consisting of all the strategies that can be executed with the following capabilities:

- The computation of any polynomial-time function (computationally bounded adversary)
- A network interface that allows eavesdropping, deletion, modification, and insertion of messages (adversary controlled network)
- The capability to corrupt any network party that is not currently executing the protocol; the corrupted party becomes an adversary (insider adversary)

The class $\mathbf{I}_0 \subset \mathbf{I}$ are those strategies in $\mathbf{I}$ for which the dependency assumption holds.

The reason to define the class $\mathbf{I}_0$ as a subset of $\mathbf{I}$ is twofold. First, it allows us to state security arguments without explicitly mentioning the negligible probability of error in dependency functions. Second, it allows us to avoid a technical issue that arises in the formalization of collision resistance in a hash (and non-injective) functions [131], as explained in the following.

The dependency assumption of a D-function holds if an adversary cannot find collisions in the D-function, but, e.g., an unkeyed hash function contains a large

number of second preimages. On average, a hash function (e.g., based on SHA-256) that maps 1024 bit input to 256 bit output contains $2^{1024-256}$ input values that map to the same output. Therefore, it is quite probable that an attack strategy $\mathcal{I} \in \mathbf{I}$ exists that finds some of these collisions, but it is only the case that no one is yet able to find the attack strategy. It is merely our belief, rather than a mathematical truth, that no one will be able to find the attack strategy in the future [131]. Thus a non-injective function cannot satisfy the dependency assumption for the class $\mathbf{I}$. The definition of class $\mathbf{I}_0$ is a way to address this technical problem, and, for instance, a strategy $\mathcal{I} \in \mathbf{I}_0$ cannot be a strategy that finds second preimages in a hash function.

3.2 Dependency Graph

Informally, a weakly connected digraph is a dependency graph (D-graph) if each of its arcs corresponds to a D-function. This concept is formalized in the following definition.

Definition 3.4 (Dependency Graph) Let $\mathbf{D} = (nodes(D), arcs(D))$ be a digraph, where $nodes(D)$ is the set of nodes (vertices) and $arcs(D)$ is the set of anti-reflexive arcs[4] (edges) of $\mathbf{D}$. The digraph $\mathbf{D}$ is a dependency graph if the following conditions hold in $\mathbf{D}$:

1. (Dependency Arcs) $\forall(M_i \rightarrow M_j) \in arcs(D) : M_j = Dep(M_i, .)$, where $Dep(M_i, .)$ is a D-function.
2. (Full Connectivity) $\forall M_i \in nodes(D), \forall M_j \in nodes(D) : walk(M_i, M_j)$, where $walk(M_i, M_j)$ is the predicate representing the existence of a walk between the nodes M_i and M_j.

In the above definition, the first condition states that for each arc of a D-graph it is the case that the head of the arc is the output of a D-function and the tail of the arc is the main argument of the D-function. In the second condition, the predicate $walk(M_i, M_j)$ stands for the requirement that all pairs of dependency nodes are connected in the underlying undirected graph.

As an example of a D-graph, consider two variables M_1 and M_2 such that $M_2 = \mathcal{E}_{Pk_A}(M_1)$ (encryption of M_1). Let the D-graph on the nodes M_1 and M_2 be denoted by $\mathbf{D}$, which is defined as follows:

[4] An anti-reflexive arc does not have the same head node and tail node, i.e., $M_1 \rightarrow M_2$ implies $M_1 \neq M_2$.

$$nodes(\mathbf{D}) = \{M_1, M_2 : M_2 = \mathcal{E}_{Pk_A}(M_1)\} \tag{3.1}$$
$$arcs(\mathbf{D}) = \{M_1 \rightarrow M_2, M_2 \rightarrow M_1 : M_2 = \mathcal{E}_{Pk_A}(M_1), M_1 = \mathcal{D}_{Sk_A}(M_2)\}$$

Here, the arc $M_1 \rightarrow M_2$ is a dependency arc, which is defined by the encryption function and the public key Pk_A as an auxiliary argument, and the arc $M_2 \rightarrow M_1$ is another dependency arc, which is defined by the decryption function and the private key Sk_A as an auxiliary argument. Since the mapping between M_1 and M_2 is bijective, therefore we do not need to specify the decryption function in this D-graph.

For the auxiliary arguments of a D-function, one can use other dependency nodes or any known data (in the protocol) that is required to compute the D-function. The known data refers to the values that are known to protocol parties or can be computed during the protocol execution.

Dependency Function	Definition	Auxiliary Argument	D-arc
$M_3 = Dep_1(M_1, \mathbf{aux}_1)$	$\mathcal{H}(M_1 \| \mathbf{aux}_1)$	M_2	$M_1 \rightarrow M_3$
$M_3 = Dep_2(M_2, \mathbf{aux}_2)$	$\mathcal{H}(\mathbf{aux}_2 \| M_2)$	M_1	$M_2 \rightarrow M_3$
$M_7 = Dep_3(M_3, \mathbf{aux}_3)$	$\mathcal{H}(\mathbf{aux}_3 \| M_3)$	M_4	$M_3 \rightarrow M_7$
$M_6 = Dep_4(M_3, \mathbf{aux}_4)$	$\mathcal{H}(M_3 \| \mathbf{aux}_4)$	M_5	$M_3 \rightarrow M_6$
$M_7 = Dep_5(M_4, \mathbf{aux}_5)$	$\mathcal{H}(M_4 \| \mathbf{aux}_5)$	M_3	$M_4 \rightarrow M_7$
$M_6 = Dep_6(M_5, \mathbf{aux}_6)$	$\mathcal{H}(\mathbf{aux}_6 \| M_5)$	M_3	$M_5 \rightarrow M_6$

Table 3.1: Dependency Relations of D-Graph of Fig. 3.1

A more detailed example of a D-graph is shown in Fig. 3.1. In Fig. 3.1-(a), functional relations between seven messages are shown, which are based on a hash function $\mathcal{H}$. These relations are $M_3 = \mathcal{H}(M_1 \| M_2)$, $M_6 = \mathcal{H}(M_3 \| M_5)$, and $M_7 = \mathcal{H}(M_4 \| M_3)$. In Fig. 3.1-(b), the corresponding dependency graph is shown, in which arcs are dependency functions. These D-functions are listed in Table 3.1.

3.2.1 D-graph Relations

Typically, a D-graph $\mathbf{D}$ that consists of all possible arcs, constructed using all possible D-functions between the nodes, contains a large amount of redundancies. It will be useful if there exist a (structurally) simpler D-graph $\mathbf{D}'$ that

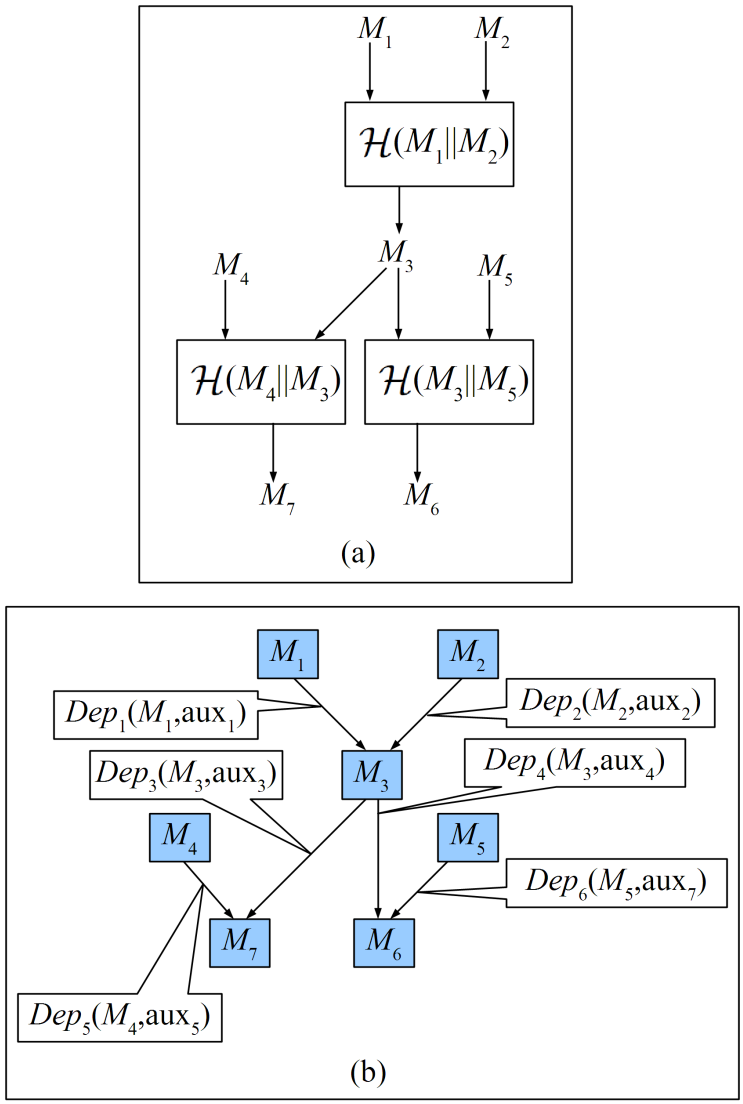


Figure 3.1: Example: (a) Functional Relations (b) Dependency Graph

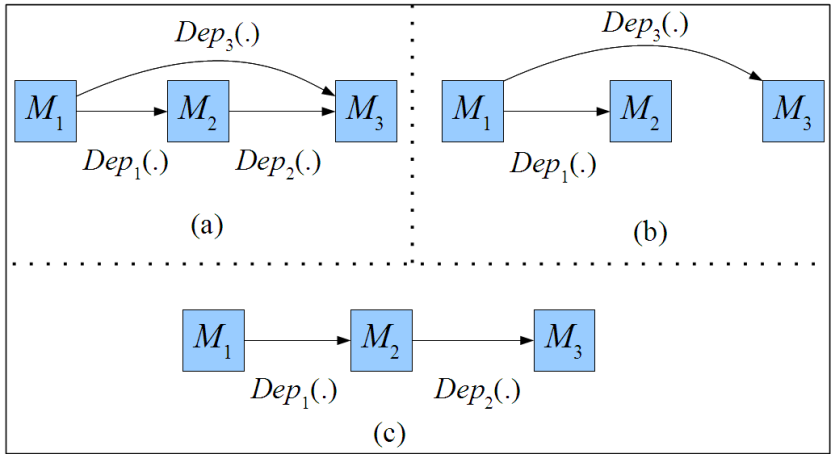


Figure 3.2: Transitive Equivalence

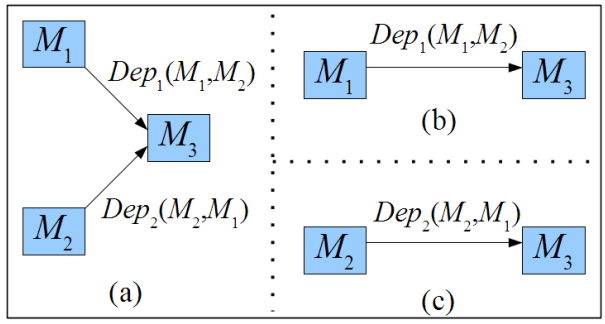


Figure 3.3: Branch Equivalence

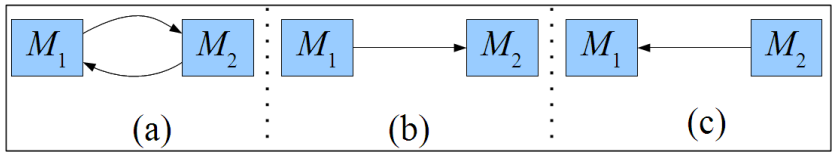


Figure 3.4: Reflexive Equivalence

represents all the dependency relations of $\mathbf{D}$. Therefore, graph equivalence is important to make further analysis easier.

First we define arc level equivalences that can be used to make two equivalent D-graphs equal. There are three types of such equivalences, which are as follows:

1. The first type is a *transitive equivalence*, which is defined as follows. Let M_1, M_2 and M_3 be three nodes of a D-graph.

$$\begin{aligned}\{M_1 \rightarrow M_2, M_2 \rightarrow M_3, M_1 \rightarrow M_3\} \equiv &\{M_1 \rightarrow M_2, M_2 \rightarrow M_3\} \\ \equiv &\{M_1 \rightarrow M_3, M_2 \rightarrow M_3\} \\ \equiv &\{M_1 \rightarrow M_2, M_1 \rightarrow M_3\} \end{aligned} \tag{3.2}$$

   To understand why a transitive equivalence holds, consider Fig. 3.2, where three D-graphs are shown, which are defined by the D-functions $M_2 = Dep_1(M_1, \mathbf{aux}_1)$, $M_3 = Dep_2(M_2, \mathbf{aux}_2)$, and $M_3 = Dep_3(M_1, \mathbf{aux}_3)$. The D-function $Dep_3(M_1, \mathbf{aux}_3)$ can be constructed by using the other two D-functions $Dep_1(.)$ and $Dep_2(.)$, and by defining $\mathbf{aux}_3 = \mathbf{aux}_1, \mathbf{aux}_2$. Any other construction of $Dep_3(.)$ is merely another way of computation, but it does not change the mapping between M_1 and M_3. Therefore, not drawing either $M_2 \rightarrow M_3$ or $M_1 \rightarrow M_3$ does not change the dependency information expressed by these graphs. Therefore, we say that these three D-graphs as equivalent D-graphs.

   Example: Let us consider three variables $M_1 = N_A$, $M_2 = \mathcal{E}_{K_1}(M_1)$, and $M_3 = \mathcal{E}_{K_2}(M_2)$. The D-functions corresponding to $M_1 \rightarrow M_2$, $M_2 \rightarrow M_3$, and $M_1 \rightarrow M_3$ are $M_2 = Dep_1(M_1, K_1)$, $M_3 = Dep_2(M_2, K_2)$, and $M_3 = Dep_3(M_1, K_1, K_2)$. Clearly, any two of these three D-functions involve the computation of the two original encryption functions. Therefore, a third D-function is redundant.

2. The second equivalence is called a *branch equivalence*. Once again, let M_1, M_2 and M_3 be three nodes of a D-graph. A *branch equivalence* is defined as follows:

$$\begin{aligned}&\{M_1 \rightarrow M_3, M_2 \rightarrow M_3\} \Rightarrow \\ &\{M_1 \rightarrow M_3, M_2 \rightarrow M_3\} \equiv \{M_1 \rightarrow M_3\} \equiv \{M_2 \rightarrow M_3\}\end{aligned} \tag{3.3}$$

   In other words, if $\{M_1 \rightarrow M_3, M_2 \rightarrow M_3\}$ exists then, e.g., $\{M_1 \rightarrow M_3\}$ and $\{M_2 \rightarrow M_3\}$ are equivalent. It is a pre-condition that $\{M_1 \rightarrow$

$M_3, M_2 \rightarrow M_3\}$ exists, because this ensures that, e.g., the D-function of $M_1 \rightarrow M_3$ also contains M_2 in its auxiliary arguments.

To understand why this equivalence holds, consider Fig. 3.3, where we have $M_3 = Dep_1(M_1, \mathbf{aux}_1 = M_2)$ and $M_3 = Dep_1(M_2, \mathbf{aux}_2 = M_1)$. The auxiliary arguments may contain other variables, but that will not affect the branch equivalence. These two D-functions only differ in the ordering of their arguments. The two arcs $M_1 \rightarrow M_3$ and $M_2 \rightarrow M_3$ have the same underlying D-function. Therefore, the three D-graphs shown in Fig. 3.3 are equivalent.

3. The third equivalence is a *reflexive equivalence*, which is defined as follows:

$$\begin{aligned} &\{M_1 \rightarrow M_2, M_2 \rightarrow M_1\} \Rightarrow \\ &\{M_1 \rightarrow M_2, M_2 \rightarrow M_1\} \equiv \{M_1 \rightarrow M_2\} \equiv \{M_2 \rightarrow M_1\} \end{aligned} \tag{3.4}$$

This equivalence is shown in Fig. 3.4, where we have $M_2 = Dep_1(M_1, \mathbf{aux}_1)$ and $M_1 = Dep_2(M_2, \mathbf{aux}_2)$, such that $Dep_2(.)$ is the inverse function of $Dep_1(.)$. Clearly, both dependency functions represent the same injective mapping between M_1 and M_2. The verification of any one of the functions implies the verification of the other function. Therefore, three D-graphs shown in Fig. 3.4 are equivalent.

Definition 3.5 (D-graph Equivalence) Two D-graphs $\mathbf{D}$ and $\mathbf{D}'$ are equivalent, denoted by $\mathbf{D} \equiv \mathbf{D}'$, if both can be made equal by applying transitive (Equation 3.2), branch (Equation 3.3), or reflexive (Equation 3.4) equivalences.

Clearly, $\mathbf{D} \equiv \mathbf{D}'$ implies $nodes(\mathbf{D}) = nodes(\mathbf{D}')$, but $\mathbf{D} \equiv \mathbf{D}'$ does not imply $arcs(\mathbf{D}) = arcs(\mathbf{D}')$. For example, the D-graph $\mathbf{D}^2$ in Fig 3.6-(a) is equivalent to $\mathbf{D}^1$ in Fig 3.5, and both D-graphs have the same set of nodes but they have different set of arcs. Similarly, for our earlier example in Equation 3.1, we have the following equivalence relations among arc sets.

$$\begin{aligned} nodes(\mathbf{D}) =& \{M_1, M_2 : M_2 = \mathcal{E}_{Pk_A}(M_1)\} \\ arcs(\mathbf{D}) =& \{M_1 \rightarrow M_2, M_2 \rightarrow M_1 : M_2 = \mathcal{E}_{Pk_A}(M_1), M_1 = \mathcal{D}_{Sk_A}(M_2)\} \\ \equiv& \{M_1 \rightarrow M_2 : M_2 = \mathcal{E}_{Pk_A}(M_1)\} \\ \equiv& \{M_2 \rightarrow M_1 : M_1 = \mathcal{D}_{Sk_A}(M_2)\} \end{aligned}$$

We use the notation $equiv(\mathbf{D})$ to denote the set of all D-graphs that are equivalent to the D-graph $\mathbf{D}$. A D-graph is always equivalent to itself, i.e., $\mathbf{D} \in$

$equiv(\mathbf{D})$.
Next, we define *subgraph* and *proper subgraph* relations[5].

$$\mathbf{D} \subseteq \mathbf{D}' \stackrel{\text{def}}{=} \exists \mathbf{D}_e \in equiv(\mathbf{D}), \exists \mathbf{D}'_e \in equiv(\mathbf{D}') : \\ nodes(\mathbf{D}_e) \subseteq nodes(\mathbf{D}'_e) \wedge arcs(\mathbf{D}_e) \subseteq arcs(\mathbf{D}'_e) \tag{3.5}$$

Similarly the proper subgraph relation is as follows:

$$\mathbf{D} \subset \mathbf{D}' \stackrel{\text{def}}{=} \exists \mathbf{D}_e \in equiv(\mathbf{D}), \exists \mathbf{D}'_e \in equiv(\mathbf{D}') : \\ nodes(\mathbf{D}_e) \subset nodes(\mathbf{D}'_e) \wedge arcs(\mathbf{D}_e) \subset arcs(\mathbf{D}'_e) \tag{3.6}$$

A D-graph $\mathbf{D}$ is a subgraph of $\mathbf{D}'$ if an equivalent D-graph of $\mathbf{D}'$ contains all of the arcs and nodes of $\mathbf{D}$. For example, we have $\mathbf{D}^3 \subseteq \mathbf{D}^2$ in Fig 3.6, which also happens to be a proper subgraph, i.e., $\mathbf{D}^3 \subset \mathbf{D}^2$.
Next, we define a maximal D-graph.

Definition 3.6 (Maximal D-graph) A maximal D-graph is a D-graph that is not a proper subgraph of any other D-graph for a given set of nodes.

Note that a maximal D-graph is a subgraph of all of its equivalent D-graph, which means that the following predicate is true for a maximal D-graph $\mathbf{D}$.

$$\forall \mathbf{D}' : \mathbf{D} \subseteq \mathbf{D}' \Rightarrow \mathbf{D} \equiv \mathbf{D}' \tag{3.7}$$

That is to say, for each D-graph $\mathbf{D}'$ of which $\mathbf{D}$ is a subgraph, it must be the case that the two D-graphs are equivalent. In simple words, there may be more than one maximal D-graphs, but all of them are equivalent.

Definition 3.7 (Distinct D-graphs) Two D-graphs $\mathbf{D}$ and $\mathbf{D}'$ are distinct D-graphs only if they do not have any node in common, i.e.,
$nodes(\mathbf{D}) \cap nodes(\mathbf{D}') = \phi$.

For example, in Fig. 3.6, $\mathbf{D}^4$ and $\mathbf{D}^5$ are distinct D-graphs, because they do not share any node.

[5]By borrowing the terminology from the set theory: subset and proper (strict) subset

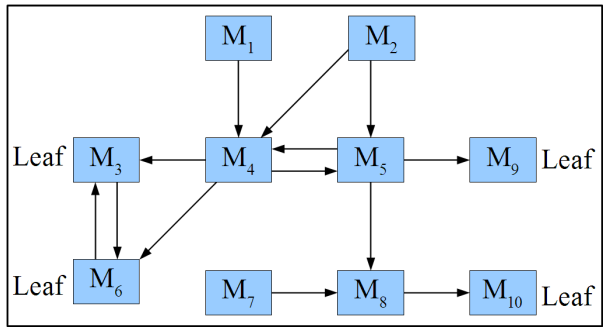


Figure 3.5: Example of a D-graph: $\mathbf{D}^1$

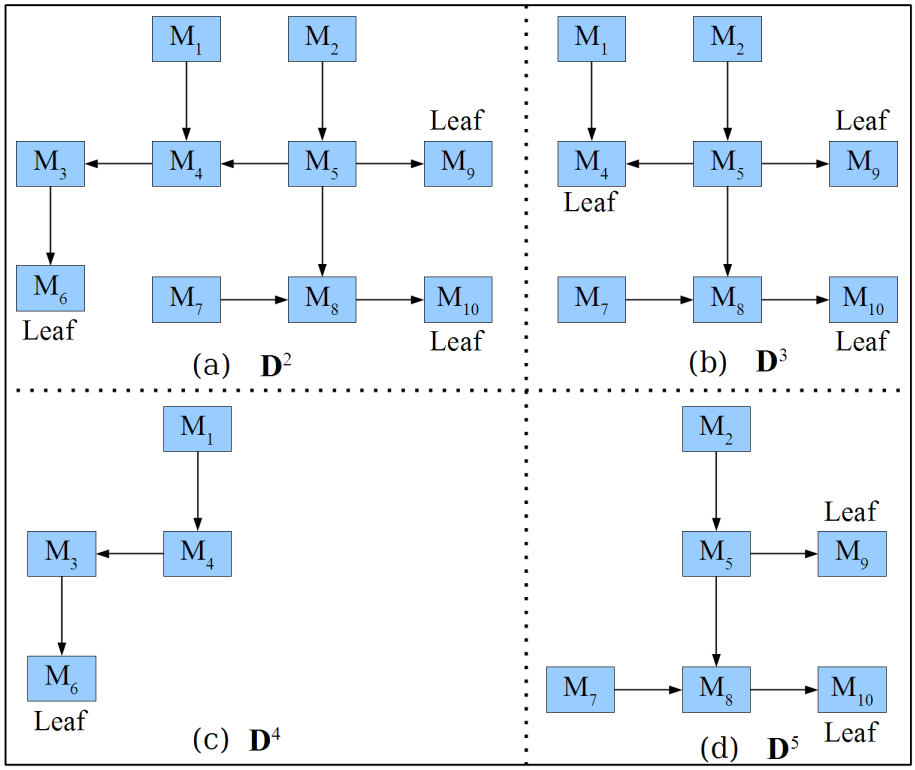


Figure 3.6: Examples of D-graphs: (a) $\mathbf{D}^2$, (b) $\mathbf{D}^3$, (c) $\mathbf{D}^4$, (d) $\mathbf{D}^5$

Definition 3.8 (Leaf Nodes) Let $\mathbf{D}$ be a D-graph and the predicate $path(M_i, M_j)$ is true if $\{M_i, M_j\} \subseteq nodes(\mathbf{D})$ and there is a directed path from M_i to M_j following dependency arcs. A set of leaf nodes, $leaf(\mathbf{D})$, is a set with the minimal order for which the following predicate is true:

$$\forall M_i \in nodes(\mathbf{D}),\ \exists M_j \in leaf(\mathbf{D}) : path(M_i, M_j)$$

That is to say, for each node of a D-graph there exist a leaf node such that there is a directed path from the node to the leaf node. Each node of a D-graph can reach some leaf node by following a directed path. We note that the requirement of minimal order implies the following:

- A node with no outgoing arc is a leaf node.
- Any node on a closed directed cycle is a leaf node. A closed directed cycle is a path that does not have an exiting arc, i.e., starting from any node on a closed cycle one always traverses back to the same node. We only need to choose one node on a directed cycle, because multiple leaf nodes on the same directed cycle are redundant.

Depending on which node of a closed directed cycle is selected as a leaf node, there may be more than one different sets of leaf nodes (with the same minimal order). For examples, in Fig. 3.5, we have $leaf(\mathbf{D}^1) = [M_3, M_9, M_{10}]$ or $leaf(\mathbf{D}^1) = [M_6, M_9, M_{10}]$. Although a set of leaf nodes is not unique, this does not pose any problem in the structured intuition (SI), and each set of leaf nodes can be used with the same results.

The intuition behind the definition of leaf nodes is that a security property, which we call canonicity and define in the next chapter, propagates on a D-graph in reverse direction. This means that if one validates the canonicity of the leaf nodes of a D-graph then the canonicity holds for all nodes of the D-graph. This reduces the amount of effort required in an SI based security analysis.

3.3 Atomicity of a D-graph

Now, we present a property that holds on a D-graph in the presence of any adversarial strategy that belongs to the class $\mathbf{I}_0$. We write $\mathbf{msg} \in nodes(\mathbf{D})$ to represent that the nodes of $\mathbf{D}$ are assigned with a list of values $\mathbf{msg}$ such that the dependency functions of $\mathbf{D}$ can be verified with the resultant values.

Proposition 3.9 (D-Graph Atomicity) *Let* $\mathbf{D}$ *be a D-graph. There is no* $\mathcal{I} \in \mathbf{I}_0$ *that computes* $\mathbf{msg}_1$ *and* $\mathbf{msg}_2$*, where* $\mathbf{msg}_1 \neq \mathbf{msg}_2$*, such that a set of*

leaf nodes of $\mathbf{D}$ *have the same value in the two assignments* $\mathbf{msg}_1 \in nodes(\mathbf{D})$ *and* $\mathbf{msg}_2 \in nodes(\mathbf{D})$.

PROOF. Using a contrapositive argument, we show that if the proposition does not hold then an adversary can generate collisions in a D-function. Since we assume that an adversary $\mathcal{I} \in \mathbf{I}_0$ cannot find collisions in a D-function, we conclude that the proposition holds.

Let us assume that $\mathcal{I} \in \mathbf{I}_0$ computes $\mathbf{msg}_1 \in \mathbf{D}$ and $\mathbf{msg}_2 \in \mathbf{D}$ such that the leaf nodes have the same values. Since $\mathbf{msg}_1 \neq \mathbf{msg}_2$, there exists a non-leaf node $M_x \in nodes(\mathbf{D})$ that has different values in $\mathbf{msg}_1$ and $\mathbf{msg}_2$. By the definition of leaf messages, there is at least one directed path starting from M_x to a leaf node M_l: $M_x \rightarrow \cdots \rightarrow M_l$.

Traversing on the path backwards, let M_l' be the tail of an arc whose head is M_l, i.e., $M_l' \rightarrow M_l$ or $M_l = Dep_l(M_l', .)$. If M_l' has different values in $\mathbf{msg}_1$ and $\mathbf{msg}_2$ then this means $Dep_l(.)$ is not a dependency function. Therefore, for the class $\mathbf{I}_0$, M_l' gets the same value in both $\mathbf{msg}_1$ and $\mathbf{msg}_2$.

Going backwards, consider M_l' as the head of an arc, and let a message M_l'' be the tail of the arc: $M_l'' \rightarrow M_l' \rightarrow M_l$. We can apply the previous arguments once again, to conclude that M_l'' gets the same value in $\mathbf{msg}_1$ and $\mathbf{msg}_2$. Repeating these steps all along the path, eventually we reach M_x, thus concluding that M_x has the same value in both $\mathbf{msg}_1$ and $\mathbf{msg}_2$.

This conclusion contradicts with the premise that the node M_x is on the path, because M_x must get different values in $\mathbf{msg}_1$ and $\mathbf{msg}_2$. Hence, the proposition holds for all strategies in $\mathbf{I}_0$. □

The above proposition can be applied to any subgraph of a D-graph. To get an intuitive sense for the relevance of Proposition 3.9 to entity authentication, note that the collision-resistance of a hash chain is a special case of Proposition 3.9. In Lamport's password authentication scheme [93], which is similar to S/Key [86], the atomicity of the hash chain is a necessary condition. In a similar way, the atomicity of a D-graph is a necessary condition to conclude various entity authentication goals, as we explain in the following chapters.

3.4 Protocol Narration as a D-Graph

The theory of a D-graph, as introduced in the previous section, is quite independent from the notion of a protocol. In fact, one can define a D-function, a D-graph, and the associated notions on any set of data variables, and whether these variables represent communication messages is not important. We consider the model of a D-graph as a modelling tool, which we believe can be applied in other applications. For instance, a hash chain, which is a special D-graph, is a cryptographic tool that can be used beyond authentication protocols. Nevertheless, the main reason for introducing a D-graph in this thesis is to model an authentication protocol.

A D-graph is a graphical representation of the dependency functions that exist among protocol messages. To distinguish between the D-graphs of a protocol and a role program (presented in the next chapter), we respectively refer to them as a global D-graph (for a protocol narration) and a local D-graph (for a role narration). In these D-graphs, most of the nodes represent message variables, but the rest of the nodes represent, what we call, interim variables:

Definition 3.10 (Interim Variable and Interim Node) Let M_1 and M_2 be two message variables, such that neither M_1 depends on M_2 nor M_2 depends on M_1. If there exist a variable M_3, where $M_3 \in tr(M_1) \cup tr(M_2)$, such that both M_1 and M_2 depends on M_3 then M_3 is called an interim variable and the corresponding node in the D-graph is called an interim node.

For example, let the two message variables be $M_1 = \mathcal{E}_{Pk_A}(R_A, R_B)$ and $M_2 = \mathcal{E}_{Pk_C}(R_C, R_B)$. Clearly, neither M_1 depends on M_2 nor M_2 depends on M_1, but both M_1 and M_2 depends on R_B. In this case, we can invent an interim variable $M_3 = R_B$. Now, we can construct a D-graph on M_1 and M_2 with the arc set $\{M_3 \rightarrow M_1, M_3 \rightarrow M_2\}$.
We define a global D-graph for a protocol narration Π as follows.

Definition 3.11 (Global D-graph) A D-graph $\mathbf{D}$ is a global D-graph of a protocol narration Π if

- $mv(\Pi) \subseteq nodes(\mathbf{D})$, i.e., all message variables in the protocol narration are contained in the node set of $\mathbf{D}$.
- $nodes(\mathbf{D}) \setminus mv(\Pi)$ is a set of interim nodes of $mv(\Pi)$.[6]

[6]Here, $nodes(\mathbf{D}) \setminus mv(\Pi)$ is the standard subtraction on sets, which is defined as $\{M : M \in nodes(\mathbf{D}) \wedge M \notin mv(\Pi)\}$.

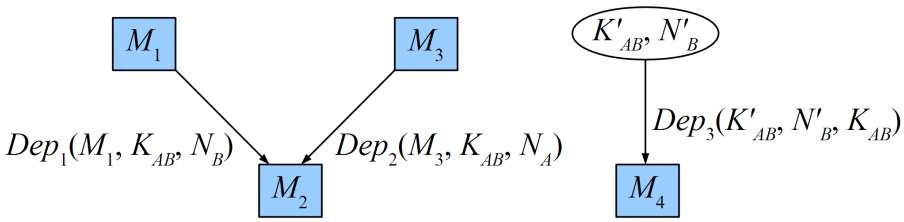


Figure 3.7: Global D-Graphs of Andrew Secure RPC Protocol

As we know, a message variable is a variable that corresponds to a received or sent message in a protocol. A global D-graph spans all of the message variables of the protocol narration and a number of interim nodes. The main reason to introduce the interim nodes is to achieve complete connectivity in a D-graph. An interim node helps in creating a walk between a pair of dependency nodes that are otherwise not connected. A global D-graph is always a maximal D-graph with respect to the message variables of the protocol, because it contains all of the message variables. The above definition allows multiple global D-graphs for a protocol narration, but all the the D-graphs must be equivalent, namely they all have the same sets of nodes.

As an example of a D-graph, consider a narration of the Andrew secure RPC protocol [136].

$$
\begin{array}{lll}
F_1 \stackrel{\text{def}}{=} & A^\rho \longrightarrow B^\rho : & A, M_1 = \mathcal{E}_{K_{AB}}(N_A) \\
F_2 \stackrel{\text{def}}{=} & B^\rho \longrightarrow A^\rho : & M_2 = \mathcal{E}_{K_{AB}}(N_A + 1 \| N_B) \\
F_3 \stackrel{\text{def}}{=} & A^\rho \longrightarrow B^\rho : & M_3 = \mathcal{E}_{K_{AB}}(N_B + 1) \\
F_4 \stackrel{\text{def}}{=} & B^\rho \longrightarrow A^\rho : & M_4 = \mathcal{E}_{K_{AB}}(K'_{AB} \| N'_B)
\end{array}
$$

The narration consists of two role programs A^ρ and B^ρ executed by two network parties A and B respectively. The function $\mathcal{E}_{K_{AB}}(.)$ represents a symmetric encryption function using a long term secret key K_{AB} shared between A and B. The terms N_A and N_B are the nonces that are generated by A and B respectively.

The authentication goals of the protocol are not relevant in the dependency analysis. We are only interested in the dependency functions that exist among the message variables: M_1, M_2, M_3, and M_4. We do not use an identity that appears in plaintext for the construction of a D-graph, therefore, the term A in the first flow is not used.

The two global D-graphs corresponding to the above narration are shown in Fig. 3.7. The first D-graph is on the first three message variables and can be constructed without introducing an interim node. The message variable M_4 is a dependency node defined by a D-arc that originates from interim variables; the variables K'_{AB}, N'_B are interim variables because they do not directly correspond to any communication messages.

Note that, unlike the key K'_{AB}, we do not use K_{AB} as an interim node to connect the two D-graphs. This is because an arc from K_{AB} to any of the other message variables does not correspond to a D-function. For instance, $K_{AB} \rightarrow M_1$ is not a valid dependency arc, because one can freely change the value of K_{AB} without changing a fixed value of M_1. Let us say we fix the value of M_1 to m_1. Now, different values for N_A and K_{AB} can be computed: $N_A = \mathcal{D}_{K_{AB}}(m_1)$. Since collisions are possible, the mapping from K_{AB} to M_1 does not meet the requirements of a dependency function.

Similarly, there is no D-arc from $M_2 = \mathcal{E}_{K_{AB}}(N_A + 1 \| N_B)$ to $M_3 = \mathcal{E}_{K_{AB}}(N_B + 1)$, because M_3 does not depend on N_A and, therefore, one can easily create collisions by simply changing the value of N_A. Intuitively, an arc $M_3 \rightarrow M_2$ means that all parts of M_3 are used to compute M_2, which is not the case in the reverse direction.

As per Def. 3.11, a protocol must correspond to a connected digraph. If a "protocol" corresponds to multiple (disconnected) D-graphs, it effectively becomes a *collection of protocols* in our methodology. Therefore, Andrew secure RPC "protocol" is a collection of two protocols, i.e., F_1, F_2, F_3 is one protocol and F_4 is another protocol. As we explain later, multiple D-graphs essentially means multiple binding sequences and each of the binding sequence is used independently to derive authentication properties.

When identifying the D-functions in a protocol narration, one must be careful not to form vacuous D-functions. For example, if a message variable M_1 is to be assigned with a random value then we can always invent another interim variable M_2 such that M_2 is the decryption of M_1 with a certain key. Clearly, now we can draw dependency arcs between M_1 and M_2. Indeed, we can continue this process to form a infinitely large D-graph. To avoid this problem, we require that one must not invent new variables that are not already present in a protocol narration.

Session

By definition, each message variable of a protocol narration is a node of a global D-graph. In an execution of the protocol, sent and received messages (i.e., values) are assigned to message variables: $mv(\Pi) \leftarrow \mathbf{msg}$, where **msg** is the transcript of the sent and received messages. If the D-functions of **G** hold on the assigned values then the assigned list of values is an instance of **G** $\mathbf{msg} \in nodes(\mathbf{G})$.

An instance of a global D-graph is also referred to as a *session* of the protocol. A session is a legal transcript of a protocol execution, because the membership of the global D-graph means that the protocol functions are satisfied with the values that occur in the session. In an operational sense, a session of an m-role protocol is generated in an execution in which m role programs communicate with each other and then they terminate successfully.

It is important to note that a session is just a list of values that *can* occur in an execution of the protocol. A session *does not* mean that such an execution has actually happened on the network between honest parties. For example, in Andrew secure RPC protocol, if we assume that an adversary knows the shared key K_{AB} then this assumption does not affect the shape or verification of the global D-graphs shown in Fig. 3.7. With this assumption, a session of the protocol can be generated by an adversary, but even the adversary generated session is a legal transcript that can occur (may be in the future) between honest parties. In particular, the notion of a session (or the membership of a D-graph) does not capture the security aspect of a protocol that provides protection against a network adversary. Further discussion is in the next chapter, where we introduce security requirements.

Conventions for D-graphs

There are many ways in which the construction process of a D-graph can be somewhat standardized. As the analysis of all equivalent D-graphs provide the same results, we adopt a few conventions for this purpose:

- (Direction of Arcs) The direction of an arc should follow the execution of a protocol in the use of a reflexive equivalence.
- (Nearest Neighbours) Dependency arcs should be drawn between the nearest neighbours if an argument occurs in two different previous flows in the use of a transitive equivalence. For instance, if $M_1 = N_A, M_2 = \mathcal{E}_K(N_A)$

and $M_3 = \mathcal{E}_K(N_A + 1)$ represent first, second and third flows of a protocol respectively then $M_1 \rightarrow M_3$ is not drawn, because the nearest neighbour from where N_A can be extracted is M_2. The resultant arc set is $\{M_1 \rightarrow M_2, M_2 \rightarrow M_3\}$.

- (Plaintext Identities) In the construction of a D-graph, we do not use the identities of parties that appear in plaintext in a protocol narration.
- (Single Node) A single node is a D-graph if it is the output of a D-function and the main argument of the D-function is an input to the protocol.

The D-graphs that we draw in this thesis comply to these conventions. With these conventions, the D-graphs of most of the commonly used authentication protocols contain arcs pointing in the direction of a protocol execution, or sidewise (between messages in a flow). It is certainly not possible to always adhere to these conventions.

3.5 Case Study 1: Global D-graph

We continuing with our running example, the five-pass authentication protocol from the international standard ISO-9798 part 2 [146, 51]. The protocol narration, without optional text fields, is as follows:

(1) $A^\rho \longrightarrow B^\rho$: $M_1 = R_A$
(2) $B^\rho \longrightarrow S^\rho$: $M_2 = R'_B, M_3 = R_A, A$
(3) $S^\rho \longrightarrow B^\rho$: $M_4 = \mathcal{E}_{K_{BS}}(R'_B \| K_{AB} \| A), M_5 = \mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B)$
(4) $B^\rho \longrightarrow A^\rho$: $M_6 = \mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B), M_7 = \mathcal{E}_{K_{AB}}(R_B \| R_A)$
(5) $A^\rho \longrightarrow B^\rho$: $M_8 = \mathcal{E}_{K_{AB}}(R_A \| R_B)$

The global dependency graph is drawn in Figure 3.8. Note that an identity function is a dependency function, which is trivially collision free, therefore one can draw an arc between two occurrence of a message, namely between M_1 and M_3.

In the D-graph, K_{AB} is a protocol term (but is not a message variable); K_{AB} acts as an interim node. The purpose of the interim node K_{AB} is to create walks between message variables, M_4, M_5, and M_6.

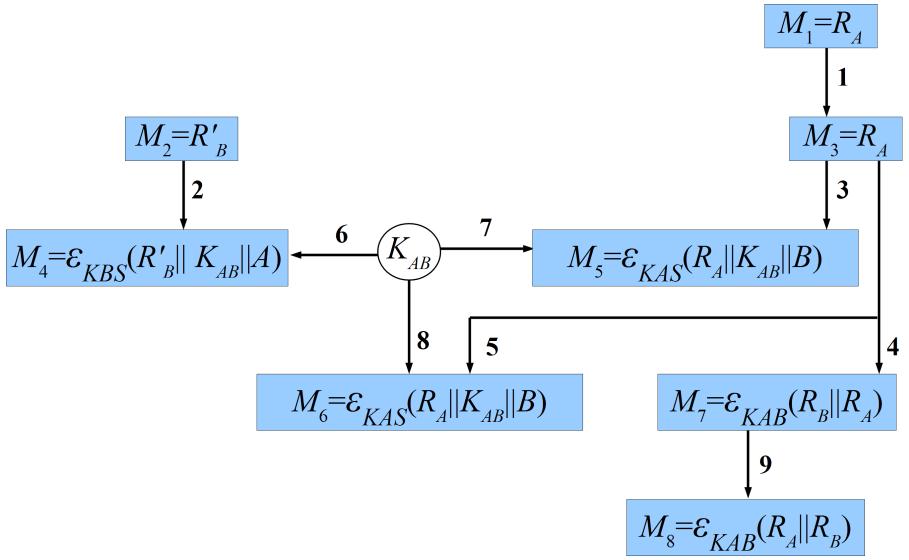


Figure 3.8: Global Dependency Graph of Working Example

Following the conventions, certain arcs are not drawn, e.g., $M_1 \rightarrow M_6$, $M_1 \rightarrow M_7$, $M_1 \rightarrow M_8$, $M_3 \rightarrow M_1$. The corresponding definitions of dependency functions are trivial and are listed below.

(1) $M_3 = Dep_1(M_1)$**:** *return* M_1

(2) $M_4 = Dep_2(M_2, K_{BS}, K_{AB}, A)$**:** *return* $\mathcal{E}_{K_{BS}}(M_2 \| K_{AB} \| A)$

(3) $M_5 = Dep_3(M_3, K_{AS}, K_{AB}, B)$**:** *return* $\mathcal{E}_{K_{AS}}(M_3 \| K_{AB} \| B)$

(4) $M_7 = Dep_4(M_3, K_{AB}, R_B)$**:** *return* $\mathcal{E}_{K_{AB}}(R_B \| M_3)$

(5) $M_6 = Dep_5(M_3, K_{AS}, K_{AB}, B)$**:** *return* $\mathcal{E}_{K_{AS}}(M_3 \| K_{AB} \| B)$

(6) $M_4 = Dep_5(K_{AB}, K_{BS}, R'_B, A)$**:** *return* $\mathcal{E}_{K_{BS}}(R'_B, K_{AB} \| A)$

(7) $M_4 = Dep_5(K_{AB}, K_{AS}, R_A, B)$**:** *return* $\mathcal{E}_{K_{AS}}(R_A, K_{AB} \| B)$

(8) $M_6 = Dep_5(K_{AB}, K_{AS}, R_A, B)$**:** *return* $\mathcal{E}_{K_{AS}}(R_A, K_{AB} \| B)$

(9) $M_8 = Dep_5(M_7, K_{AB})$**:** $x_1, x_2 = \mathcal{D}_{K_{AB}}(M_7)$;
return $\mathcal{E}_{K_{AB}}(x_2 \| x_1)$

In these D-functions, we have freely used protocol terms as auxiliary arguments, so that the D-functions can be computed in a global perspective. As mentioned in the start of this chapter, a global D-graph is not an operational model of the protocol. In an actual execution, the D-functions are computed (or verified) by protocol parties, which in this example are A, B, and S. Clearly, not all of the above D-functions can be computed by A or B alone.

The main purpose of the global perspective is to introduce the reader to the notions of a D-function, a D-graph, and a session, in a minimal setup. In the next chapter, we describe D-graphs from the perspective of a protocol party.

3.6 Summary

In this chapter, we introduced the notion of a generic dependency function (D-function). A D-function is a mapping for which it is infeasible for a computationally bounded adversary to find two different values in the input domain that map to the same value in the output domain. An injective function, such as an encryption, decryption, or identity function, is a D-function. A non-injective function, such as a hash or signature function, could be a D-function depending on its cryptographic properties and the types of its arguments. We introduced an adversary class $\mathbf{I}_0$ as a set of strategies for which the dependency assumption holds. This formulation allows us to employ D-functions without explicitly mentioning their negligible probability of errors. We introduced the notion of a dependency graph (D-graph), which is a fully connected digraph, in which an arc corresponds to a D-function, the head node of the arc is the main argument of the D-function, and the tail node of the arc is the output of the D-function. Each instance of a D-graph is atomic in a sense that an adversary in $\mathbf{I}_0$ cannot find another instance such that the both instances have the same values for the leaf nodes. For example, assuming that a hash function is a D-function, no adversary in $\mathbf{I}_0$ can find two instances of the hash chain that share the leaf node. We described how a protocol narration can be modelled as a D-graph, which we refer to as a global D-graph. An instance of a global D-graph is also called a session of the corresponding protocol. At last, we modelled the five-pass protocol of ISO-9798 part 2 as a global D-graph.

Chapter 4

Binding Sequence

This chapter describes the first and second steps of the structured intuition methodology. As described earlier, a protocol is specified as a narration. A protocol narration consists of a number of role programs that communicate with each other in a series of flows. In the first step, we model each role program as a set of D-graphs.

The second step consists of specifying the security requirements and deriving the binding sequences of a role program. For this purpose, first, we define the notion of a canonical message, and elaborate on its verification aspect. In particular, we require certain messages, which roughly corresponds to the leafs of a D-graph, to be canonical messages. The verification of canonicity is the only part of our methodology that requires to consider the dynamic behaviour of a protocol in the presence of a network adversary. A binding sequences of a role program consists of received messages that are canonical and linked together through a D-graph. A binding sequence is generated in a single session, and it is used to derive different authentication properties.

4.1 D-Graph of a Role Program

A local D-graph represents the structure of message variables of a role program. The structure is determined by the dependency functions that exist among the message variables. Unlike a global D-graph, verification of the D-functions of a local D-graph is an essential requirement, which is defined as follows:

Definition 4.1 (Verification of D-function) A D-function $M_{i'} = Dep(M_i, .)$ is verifiable in j-th role program $\rho_j(.)$ if $\rho_j(.)$ can efficiently verify that functional mapping between $M_{i'}$ and M_i is consistent with that of $M_{i'} = Dep(M_i, .)$.

In a D-graph, only known values can be used as auxiliary arguments, which include the constants and received messages of a role program. The verification of a dependency function $M_{i'} = Dep(M_i, .)$ means that one can construct a function in the corresponding role program that carries out one of the following tasks.

1. The role program computes the D-function $M_{i'} = Dep(M_i, .)$ itself, e.g., $Dep(M_i, .)$ may stands for an encryption function for which the key and other required inputs are known in the role program.
2. The role program knows the required NP-certificate[1], which is then used to assert that the values of M_i and $M_{i'}$ are related by the dependency function, e.g.,
   - signature verification if $M_{i'}$ is a signature on M_i; here the public key acts as an NP-certificate.
   - verification via a trusted third party; the reply from the trusted party plays the role of an NP-certificate.

One can assume a verification function is efficient if it can be computed in a polynomial time, but the exact interpretation of efficiency is a subject of further interpretation depending on a given implementation, e.g., an efficient function on a personal computer may turns out to be infeasible on a smart card due to memory constraints. For the j-th role program, a local D-graph is defined as follows.

Definition 4.2 (Local D-graph of Role Program) A D-graph $\mathbf{D}_j$ is a local D-graph of a role program $\rho_j(.)$ if

1. $\mathbf{D}_j$ is a maximal D-graph on the message variables $mv(\rho_j(.))$.
2. the set $nodes(\mathbf{D}_j) \setminus mv(\Pi)$ consists of interim nodes of $mv(\rho_j(.))$.

[1] As previously described, a D-function is a non-deterministic polynomial time function. An NP-certificate consists of non-deterministic choices made by the function.

3. the D-functions of $\mathbf{D}_j$ are efficiently verifiable in $\rho_j(.)$.

Unlike, a global D-graph, the D-functions of a local D-graph must be verifiable by the corresponding role program; mere existence of dependency functions among messages is of no use if the role program cannot verify the dependency functions.

A local D-graph is a maximal D-graph on the message variables $mv(\rho_j(.))$ of a role program $\rho_j(.)$. As we know, a maximal D-graph is not a subgraph of any other D-graph that can be constructed on a given set of nodes. The reason for this requirement is that we want to maximize the size of a binding sequence, and the size of a binding sequence cannot be larger than the number of nodes (message variables) in a local D-graph. The larger a binding sequence is, the more authentication goals can be derived.

There may be more than one local D-graphs on $mv(\rho_j(.))$, but each of the local D-graphs is a distinct D-graph. Two distinct D-graphs do not have a common node. If two D-graphs are not distinct then they cannot qualify to be maximal, because the two D-graphs can then be combined to obtain a larger D-graph.

As an example of local D-graphs, we consider Andrew secure RPC protocol once again. Since this is a two-role protocol, the narration of each of the two role programs is the same as that of the protocol. In the initiator program, denoted by $A^\rho = \rho_1(N_A, K_{AB})$, the list of flows is as follows.

$$
\begin{array}{lll}
F_1 \stackrel{\text{def}}{=} & A^\rho \longrightarrow B^\rho : & M_1 = \mathcal{E}_{K_{AB}}(N_A) \\
F_2 \stackrel{\text{def}}{=} & B^\rho \longrightarrow A^\rho : & M_2 = \mathcal{E}_{K_{AB}}(N_A + 1 \| N_B) \\
F_3 \stackrel{\text{def}}{=} & A^\rho \longrightarrow B^\rho : & M_3 = \mathcal{E}_{K_{AB}}(N_B + 1) \\
F_4 \stackrel{\text{def}}{=} & B^\rho \longrightarrow A^\rho : & M_4 = \mathcal{E}_{K_{AB}}(K'_{AB} \| N'_B)
\end{array}
$$

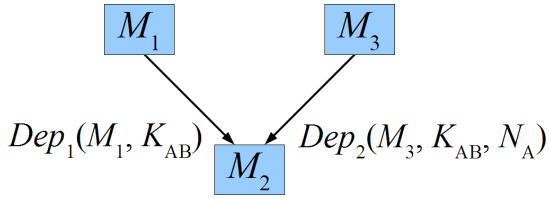


Figure 4.1: Local D-Graph of A^ρ

Unlike the global D-graphs of the protocol, in the initiator program, the message M_4 cannot be regarded as a local D-graph, because the terms K'_{AB} and N'_B are

not known to A^ρ (without using M_4 itself), and therefore A^ρ cannot verify this function[2]. This means that M_4 is not a dependency node for A^ρ. The resultant local D-graph is shown in Fig 4.1.

Note that we use a different form of the D-function for $M_1 \rightarrow M_2$, compared to the D-function used for $M_1 \rightarrow M_2$ in the global D-graph. This is because N_B is not a known value in A^ρ. The resultant function $Dep_1(.)$ is now a non-deterministic function, precisely because N_B is not known. The role program can verify the dependency relation using the decryption function. With similar arguments for the responder program B^ρ, there are two D-graphs, which are the same as the two global D-graphs shown in Fig 3.7.

The analysis of any of the equivalent local D-graphs provides the same results. We use the same conventions as we introduced for global D-graphs:

- (Direction of Arcs) The direction of an arc should follow the execution of the role program in the use of a reflexive equivalence.
- (Nearest Neighbours) Dependency arcs should be drawn between the nearest neighbours if an argument occurs in two different previous flows in the use of a transitive equivalence.
- (Plaintext Identities) In the construction of a local D-graph, we do not use the identities of parties that appear in plaintext in a role narration.
- (Single Node) A single node can be considered as a D-graph if it is the output of a D-function and its main argument is a constant of the corresponding role program.

We define the meaning of a dependency compliant run (DC-run) of a role program $\rho_j(.)$. We write $\mathbf{msg} \in nodes(\mathbf{D}_j)$ to represent that the nodes of $\mathbf{D}_j$ (message variables) are assigned values from a message list **msg** and the D-functions of $\mathbf{D}_j$ can be verified with this assignment.

Definition 4.3 (DC-Run) Let **msg** be a list of messages occurring in a run of a role program $\rho_j(.)$, whose D-graphs are in the set $\mathbf{G}_j$. The run is dependency compliant (DC) only if the predicate $\forall \mathbf{D}_j \in \mathbf{G}_j : \mathbf{msg} \in nodes(\mathbf{D}_j)$ is true.

Here, $\mathbf{G}_j$ denotes the set that consists of the D-graphs of j-th role program. In a DC-run, $\rho_j(.)$ verifies the dependency arcs of all of its D-graphs with the assignments by the messages sent and received in a run. If the verification fails then $\rho_j(.)$ outputs an error signal.

[2]In the traditional cryptographic definition [77], an encryption function is not expected to protect the integrity of a ciphertext.

As described in the previous chapter, an instance of a global D-graph represents a session of the corresponding protocol. Since a role narration is a subset of a protocol narration, an instance of a local D-graph represents a *partial session* of the protocol.

A partial session is a list of values that *can* occur during the execution of a role program while communicating with expected role programs at far-ends. Similar to a session, a partial session does not mean that this execution has already occurred. For instance, consider the previous example of the initiator role program of Andrew secure RPC protocol. A list of values for M_1, M_2, and M_3, on which $Dep_1(.)$ and $Dep_2(.)$ hold, is a partial session defined by the messages of the initiator program; as per the definition of a D-graph, this list is an instance of the local D-graph shown in Fig. 4.1. This list can occur in the initiator program while communicating with the responder program. The dependency relations, however, do not guarantee that this list has (already) occurred while communicating with the responder program. The initiator program may have communicated with an adversary to generate this partial session. To guarantee that a partial session has occurred, canonicity of messages plays an important role, which we describe in the next section.

4.2 Canonical Messages

The set of terms that appear in the message variable M_i is denoted by $tr(M_i)$, e.g., if $M_i = \mathcal{E}_{K_{AB}}(R_A \| R_B)$ then $tr(M_i) = \{K_{AB}, R_A, R_B\}$. For a protocol narration Π, the list of message variables is $mv(\Pi)$, and for $M_i \in mv(\Pi)$ the set of terms in M_i is $tr(M_i)$.
Similarly, the list of message variables in j-th role program is $mv(\rho_j(.))$, and for $M_i \in mv(\rho_j(.))$ the set of terms in M_i is $tr(M_i)$. e.g., if $M_i = \mathcal{E}_{K_{AB}}(R_A \| R_B)$ then $tr(M_i) = \{K_{AB}, R_A, R_B\}$. Clearly, each term of a role program corresponds to a term of the protocol.

To address the individual runs of a role program in a concise manner, we introduce a notational artefact. For a role program $\rho_j(.)$, let $\rho_j = \{\rho_j^1, \ldots, \rho_j^r, \ldots, \rho_j^c\}$ be a set consisting of the pointers to the previous runs in the network. Note that the set ρ_j contains the pointers to the runs from all the parties that have executed $\rho_j(.)$.

Definition 4.4 (Canonical Message) Let

1. $M_{i \in \mathbb{N}} \in \mathbf{Rx}_j$ be a variable for a received message in the role program $\rho_j(.)$.

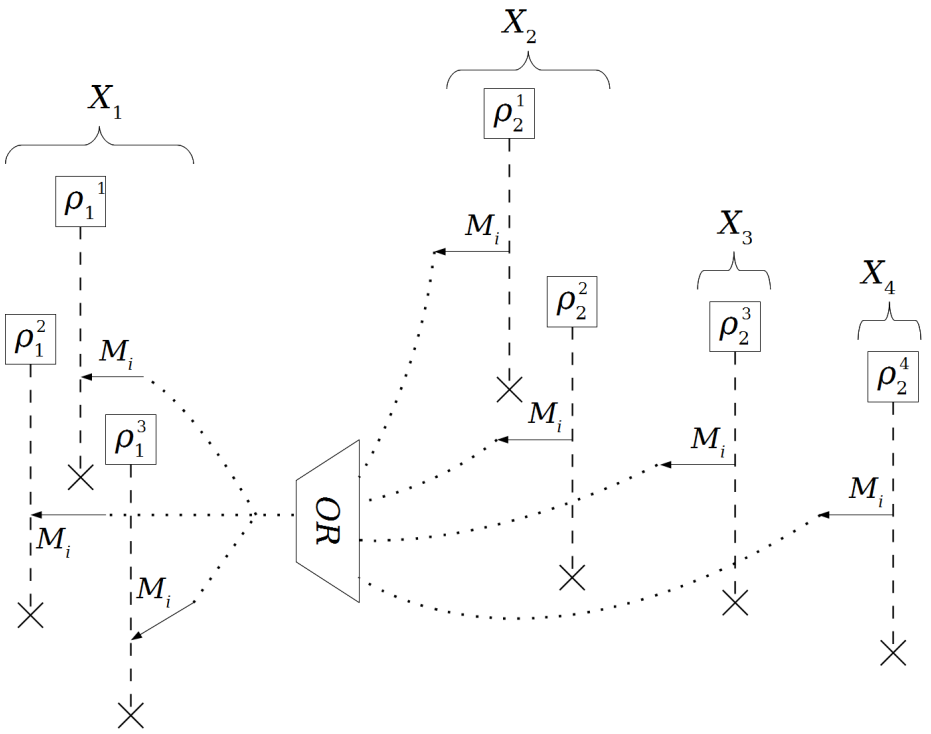


Figure 4.2: An example setup to illustrate a canonical message

2. $M_{i' \in \mathbb{N}} \in \mathbf{Tx}_{j'}$ be a variable for a sent message in the role program $\rho_{j'}(.)$ such that $tr(M_i) = tr(M_{i'})$ and $flow(M_i) = flow(M_{i'})$.

A received message msg for M_i is a canonical message in a DC-run of $\rho_j(.)$ if msg is the value of $M_{i'}$ in $\rho_{j'}^{r'}$, where $\rho_{j'}^{r'} \in \rho_{j'}$ is an existing run[3] of $\rho_{j'}(.)$ by some honest party.

We illustrate the above definition in Fig. 4.2, in which a snapshot of the execution of a two-role protocol on four honest entities X_1, X_2, X_3, and X_4 is shown. The protocol consists of two role programs $\rho_1(.)$ and $\rho_2(.)$. The snapshot contains three runs of $\rho_1(.)$ on X_1: ρ_1^1, ρ_1^2, and ρ_1^3. Similarly, there are four runs of $\rho_2(.)$, but they are on different parties.

In Fig. 4.2, each role program contains a message variable M_i that maps to the

[3]Note that we do not require $\rho_{j'}^{r'}$ to be a DC-run.

same set of terms. The program $\rho_1(.)$ receives a value from the network and assign it to a variable corresponding to M_i. The program $\rho_2(.)$ is the sender of M_i, i.e., the value of M_i is assigned in $\rho_2(.)$, and $\rho_2(.)$ sends the assigned value on the network.

As per Def. 4.4, M_i is a canonical message in a DC-run of $\rho_1(.)$ if there exist a run of $\rho_2(.)$ on the network that has previously sent this message. The inputs of *OR* block in the figure indicates that any of the runs of $\rho_2(.)$, whether it is on X_2, X_3 or X_4, satisfies our requirement. The output of *OR* block indicates that multiple runs of $\rho_1(.)$ may receive the same canonical message, thus indicating a non-injective assignment; in other words, a canonical message can be replayed in multiple runs.

A canonical message is a message on which the sender and receiver programs agree, e.g., for Andrew RPC protocol, the initiator and the responder programs must agree on the value of $M_3 = \mathcal{E}_{K_{AB}}(N_B + 1)$ if the value of M_3 is to be considered a canonical message. For a canonical message, the sender of the message and the freshness of the message are not important. That is why, the run $\rho^{r'}_{j'}$ is not bound to a party in Def. 4.4.

It is important to realise the difference between a canonical message and an authentic message[4]. In the context of a protocol, an authentic message is generated by a particular role program, in a particular flow, and by a particular honest[5] party. On a receiver of an authentic message, the origin of the message must be clear, where the origin is defined by three attributes: the identity of the party; the role program used by the party, and the flow in which the message occurs in the role program. Formally, these requirements correspond to a non-injective agreement on a message [97].

On the other hand, canonicity requires that it must be clear to a receiver of a canonical message that the message is sent by a particular role program, in a particular flow, and by some honest party. Canonicity requirement does not bind the message to a particular sender party. A comparison between authenticity and canonicity is illustrated in Fig. 4.3.

For example, let us consider two parties A and B that share a secret key K_{AB}. The parties A and B execute a two-role protocol that has only one flow: $\mathcal{E}_{K_{AB}}(T)$, where T is a time-stamp. If the receiver role program on A receives a message for $\mathcal{E}_{K_{AB}}(T)$ then A concludes that this message is either sent by A or B. The message $\mathcal{E}_{K_{AB}}(T)$ is not authentic, because it cannot be associated with one sender party, but the message is a canonical message.

[4] An authentic message provides data origin authentication.

[5] An honest party is one who executes a role program as per the protocol specification.

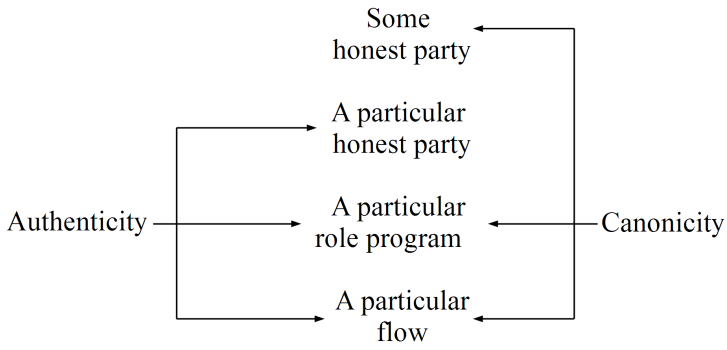


Figure 4.3: Comparison between authenticity and canonicity

The canonicity of a message is closely related to the existing notion of non-injective agreement in Lowe's authentication hierarchy [97]. The canonicity is a non-injective agreement on a message between two role programs, but without any regard to the parties that are executing these role programs. For instance, consider a situation in which Alice believes that she has completed a run while communicating with Bob, but actually Alice's run was with Bill. This situation does not prevent achieving canonicity, but this situation is not allowed for a non-injective agreement.

On the other hand, the non-injective agreement on a message implies that the message is a canonical message. Therefore, a non-injective agreement can be used as a formal requirement of canonicity in existing security models. In the running example at the end of this chapter, we use a model checker to verify non-injective agreement to conclude the canonicity of messages.

In Def. 4.4, we use the two conditions $tr(M_i) = tr(M_{i'})$ and $flow(M_i) = flow(M_{i'})$, instead of a simple but more strict condition $M_i = M_{i'}$. The condition $M_i = M_{i'}$ implies $tr(M_i) = tr(M_{i'})$ and $flow(M_i) = flow(M_{i'})$. The reason is that, in some protocols, it is trivial for an adversary to change the order of messages within a flow while playing a man-in-middle role between the two role programs. Sometimes, it does not change the agreement on the values of protocol terms ($tr(M_i) = tr(M_{i'})$) but it may change the agreement on message variables ($M_i \neq M_{i'}$). Since we derive authentication properties from the values of protocol terms and the flows in which they occur, such re-ordering "attacks" are not harmful for the purpose of entity authentication. For further explanation, consider the following two related example.

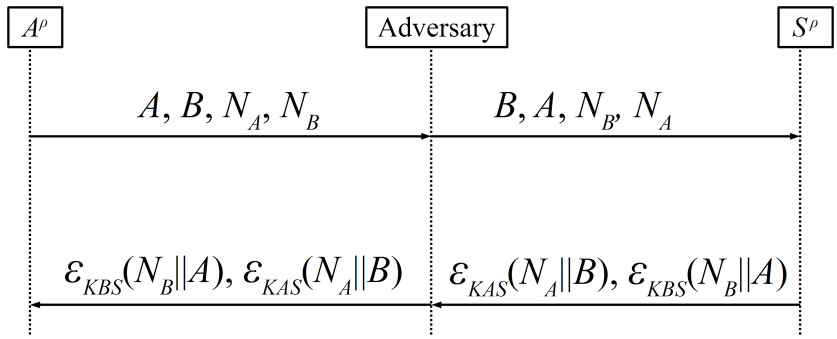


Figure 4.4: A harmless re-ordering

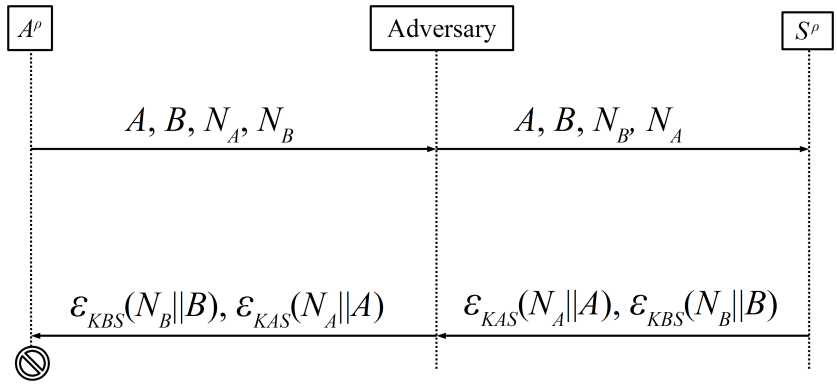


Figure 4.5: A harmful re-ordering

The first example of re-ordering is shown in Fig. 4.4. Here, a role program A^ρ executed by A sends two nonces to a role program S^ρ executed by a trusted server S. The server S shares long term keys K_{AS} and K_{BS} with A and B respectively. The first nonce N_A is A's nonce and the second nonce is B's nonce. An adversary, who controls the network, change the order of messages. This change of order does no affect the server, because it still believes that N_A is from A and N_B is from B. Therefore, both A^ρ and S^ρ agree on protocol terms as well as on the flow number.

Now, consider the second example in Fig. 4.5. Here, the re-ordering is done only between nonces. As a result, the server program S^ρ now believes that N_A is from B and N_B is from A. In other words, there is a disagreement on the protocol terms. This attack, however, cannot be mounted because A^ρ can detect

the problem during the verification of encryption function, which now contains A's identity instead of B's identity.

4.2.1 Importance of Canonicity

The decision whether an authentication goal is achieved is made on a per run basis. We say that a far-end party is acting honestly in a run if the far-end party follows the protocol and executes the expected role program. A claim of honesty of a party is only valid within the scope of a run.

There may be alternate ways to define the meaning of a far-end party being honest, e.g., one may insist that a party can be honest even if it is not following an expected role program, or honesty should be defined over all possible executions. It is not clear to us how these alternate ways can be specified and whether they are more relevant to entity authentication.

We now present a context that makes it clear why canonicity is important. Assume that a transcript of messages **msg** is received in a run of a role program $\rho_j(.)$. The role program can easily decide whether the run is dependency compliant, by verifying the D-functions of the local D-graphs of $\rho_j(.)$ with the assignments from **msg**. If the verification succeeds then $\rho_j(.)$ concludes that **msg** is a legal transcript of the protocol.

The legal transcript **msg**, however, does not imply that this transcript is the result of an execution of the protocol between honest parties, because **msg** may be computed by an adversary without following the protocol. The transcript **msg** *can* occur in an honest execution, because it satisfies the dependency relations, but one cannot conclude that whether such an execution *has already* occurred.

The notion of a D-function is too abstract to decide whether a received message is from an honest party. For example, an identity function and an encryption function, which both are dependency functions, are in fact very different functions in terms of providing protection against adversarial interference. Inevitably, we have to consider actual instantiations of a D-function. Not only the instantiations of a D-function but also the adversary capabilities and environment assumptions are important to distinguish between a message that can be sent by an honest party and a message that is actually sent by an honest party. A security analysis answers this question.

A canonical message is sent by some honest party following the protocol. A receiver may not know the sender of a canonical message, but the receiver does

know the message is not created by an adversary. In a general sense, when a canonical message is received, one concludes that the message is actually sent by some honest party. The canonicity, which is defined on a message, can be carried to a complete run using the atomicity property of a D-graph. In this way, canonicity is used to distinguish between an execution that can (possibly) occur and an execution that has actually occurred between honest parties.

4.2.2 Verification of Canonicity

In the structured intuition, we do not prescribe a general procedure for the verification of canonical messages, and a proof that a received message is a canonical message will depend on the actual protocol and method used for security analysis. For this purpose, an operational interpretation of Def. 4.4 is as follows: A message of a protocol is a canonical message if an adversary cannot compute the message without following the protocol and running the role program that computes the message.

In some examples that appear later in the thesis, we use informal arguments to claim the canonicity of a message. More rigorous proofs of canonicity can be constructed, e.g., a proof that reduces the canonicity of a message to the security of a cryptographic primitive used in the message. Some hints on how to construct proofs for the canonicity goal are as follows.

Symmetric encryption and message authentication using long term keys of a party, and signing a message using the private key of a party, are the examples of functions for which it is reasonable to assume that an adversary cannot compute these functions locally, without making use of honest parties. This assumption holds under a typical attacker model, but if a long term key can be compromised, if encryption can be broken, or if signatures can be forged then this assumption does not hold. Further, an adversary may able to use a party as an oracle to get the desired output. Therefore, it is essential to analyse a protocol for the presence of such oracles.

The security analysis of public key encryption requires a bit more attention, because all parties can compute the encryption function, including an adversary. In order to distinguish between what is sent by an adversary (not following the protocol) and what is sent by an honest party (following the protocol), a common technique is to rely on an extra protocol flow, which we call the interrogation step. The technique is similar to an authenticated recipient [79] or an authentication test [83].

In the interrogation step, an honest party A encrypts a list of message, $[M_1,$

Channel Name	Example	Meaning
Insecure Channel	$A \longrightarrow B : N_A$	No guarantee about the sender or the receiver
Confidential Channel	$A \longrightarrow B : \mathcal{E}_{Pk_B}(N_A)$	No guarantee about the sender; the receiver is B
Canonical Channel	$A \longrightarrow B : \mathcal{E}_{K_{AB}}(N_A)$	The sender is either A or B; No guarantee about the receiver
Authentic Channel	$A \longrightarrow B : N_A, \mathcal{S}_{Sk_A}(N_A)$	The sender is A; No guarantee about the receiver
Secure Channel	$A \longrightarrow B : \mathcal{E}_{K_{AB}}(A\|B\|N_A)$	The sender is A; The receiver is B

Table 4.1: Example use of cryptographic primitives

..., M_i, ..., M_n], such that at least one of those message, say M_i, is a confidential message. The ciphertext, $\mathcal{E}_{Pk_B}(M_1\|...\|M_n)$, is then sent to a party B, where Pk_B is the public key of B. If A later receives a message that is a function of this confidential message then this guarantees that B has received the earlier messages in the interrogation step. Further, at some point B has emitted a message that contains M_i.

(Interrogation)	$A^\rho \rightarrow B^\rho$:	$\mathcal{E}_{Pk_B}(M_1\|...\|M_i\|...\|M_n)$
(Reply)	$B^\rho \rightarrow A^\rho$:	M_i'

Here, the messages M_i' and M_i are related by a D-function, namely the D-arc $M_i' \rightarrow M_i$ holds, e.g., $M_i' = M_i$. Clearly, if M_i is confidential then A gets the guarantee that M_i' is indeed sent by B after decrypting $\mathcal{E}_{Pk_B}(M_1\| ...\|M_i\|$ $...\|M_n)$, but if M_i is not confidential then an adversary can compute the reply itself. A further analysis is required to reach on any additional conclusions regarding the reply. For instance, if the reply contains other messages besides M_i' than this does not necessarily mean that these messages are also sent by B.

The use of different cryptographic primitives can be described in terms of channel notions that are introduced by Maurer and Schmid [103]. These notations are summarized in Table 4.2.2. For example, we say that a nonce N_A is sent on an authentic channel by A if N_A is accompanied by the signature on N_A computed using A's private key.

To detect typing problems, we term two different messages as *type-compatible*

for a receiving party if an adversary can replace one with the other without the possibility of being detected by the party, i.e., the party recognize both of these messages as valid messages satisfying the local dependency graph. Therefore, in a security analysis, one must consider the probability of replacing a message with any of its compatible messages occurring in the protocol.

Note that replacing a message in a run with a different message from a previous run is not a typing attack if both of these messages are assigned to the same message variable. For example, in the role program B^{ρ} of Andrew secure RPC protocol, a message for M_3 can be replaced with another message for M_3. The protection against this type of replacement is guaranteed by the local dependency graph, which guarantees that the messages of the role program are interconnected.

In the following, we present a useful property of a D-graph that allows to verify the canonicity of all of the nodes by only verifying the canonicity of a sub-set of the nodes.

Proposition 4.5 (Propagation of Canonicity) *Let M_i and $M_{i'}$ be the two nodes on a path, $M_i \rightarrow \cdots \rightarrow M_{i'}$, of a local D-graph $\mathbf{D}_j$ of $\rho_j(.)$, such that $sender(M_i) = sender(M_{i'})$ and $flow(M_{i'}) \geq flow(M_i)$. In a DC-run, if the value of $M_{i'}$ is a canonical message then the value of M_i is also a canonical message, in the presence an adversary strategy $\mathcal{I} \in \mathbf{I}_0$.*

PROOF. Let msg_i and $msg_{i'}$ be the two received messages for M_i and $M_{i'}$ respectively in a run ρ_j^r of $\rho_j(.)$. The message $msg_{i'}$ is a canonical message, namely it is sent by a role program $\rho_k(.)$ in a run $\rho_k^{r'}$, where $k \neq j$. Since $flow(M_{i'}) \geq flow(M_i)$, the run $\rho_k^{r'}$ has already sent a message $msg2_i$ for M_i such that the path $msg2_i \rightarrow \cdots \rightarrow msg_{i'}$ can be verified. If $msg2_i \neq msg_i$ then it means that there is a collision in one of the dependency functions on the path. As per our assumption, no strategy in $\mathbf{I}_0$ generates such collisions. Therefore, we conclude that $msg2_i = msg_i$ and the value of M_i is a canonical message. □

For example, in a hash chain the canonicity of the leaf node is enough to conclude the canonicity of the rest of of the nodes. Note that the condition $flow(M_{i'}) \geq flow(M_i)$ is necessary because $\rho_k^{r'}$ may not be a DC-run, e.g., if $M_{i'}$ occurs in an earlier flow then ρ_j^r cannot conclude that msg_i was once sent by $\rho_k^{r'}$. Due to $flow(M_{i'}) \geq flow(M_i)$, we know that msg_i was once sent by $\rho_k^{r'}$.

The canonicity is not propagated in a forward direction, e.g., if M_1 is a canonical message in a dependency arc $M_1 \rightarrow M_2$ then M_2 may not be a canonical

message. This can be seen by considering the underlying dependency function: $M_2 = Dep(M_1, \mathbf{aux}_1)$. By changing the value of $\mathbf{aux}_1$, an adversary may able to change the value of M_2 for the same value of M_1.

4.3 Binding Sequence

As described earlier, a partial session of a protocol is an instance of a local D-graph of the protocol. A partial session is a list of values that can occur when honest parties execute the protocol. A binding sequence provides a guarantee to a role program that the partial session has already occurred. This is essentially achieved by requiring that each message in a partial session is a canonical message.

Definition 4.6 (Binding Sequence) A binding sequence β_j is a list of canonical messages in a run of $\rho_j(.)$, such that all of the canonical messages have occurred in a single partial session.

A sublist of a binding sequence is also a binding sequence. A binding sequence, which is defined on a list of messages, is an extension of canonicity requirement, which is defined on one message. A canonical message is also a binding sequence of length one, but a list consisting of more than one canonical messages may not be a binding sequence, because these canonical messages may belong to different partial sessions.

A binding sequence is generated in a single execution of a protocol. The messages of a binding sequence are sent by parties acting honestly in the protocol flows corresponding to these messages. Each message in a binding sequence is sent by a certain role program and for a certain message variable. For instance, in a two-role protocol, a binding sequence in the initiator program is the list of messages sent by a single execution of the responder program.

In Fig. 4.6, we extend our earlier example and elaborate on the meaning of a binding sequence with a snapshot of execution of a two-role protocol. The two role programs are $\rho_1(.)$ and $\rho_2(.)$. Three runs of $\rho_1(.)$ are shown in the left side of the figure; these runs occur on a party X_1. Similarly, on the right side of the figure, four runs of $\rho_2(.)$ are shown, which occur on the parties X_1, X_2 and X_3.

The role program $\rho_2(.)$ is the sender of a list of messages that are assigned to M_1, M_2, M_3, and M_4, and the program $\rho_1(.)$ is the corresponding receiver. The OR-block in the middle indicates that any of the four paths on the right can be connected to any of the three paths on the left. Each such path leads to a

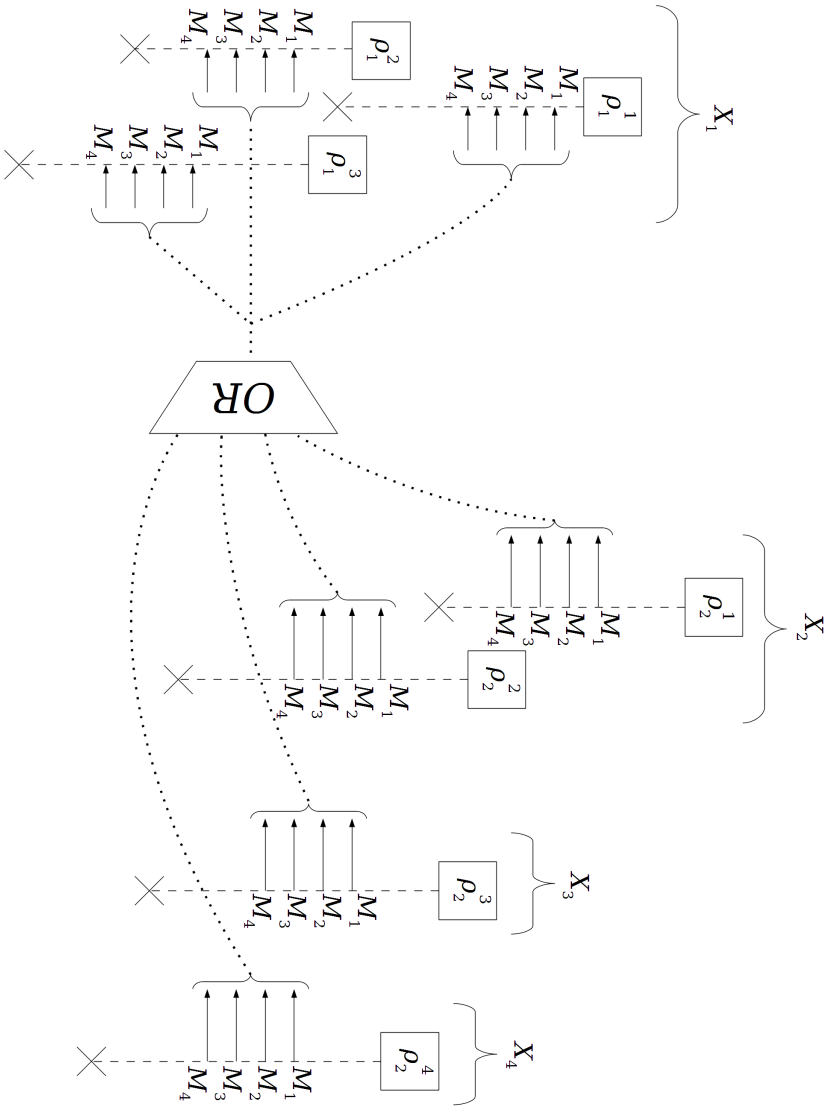


Figure 4.6: Example of a binding sequence

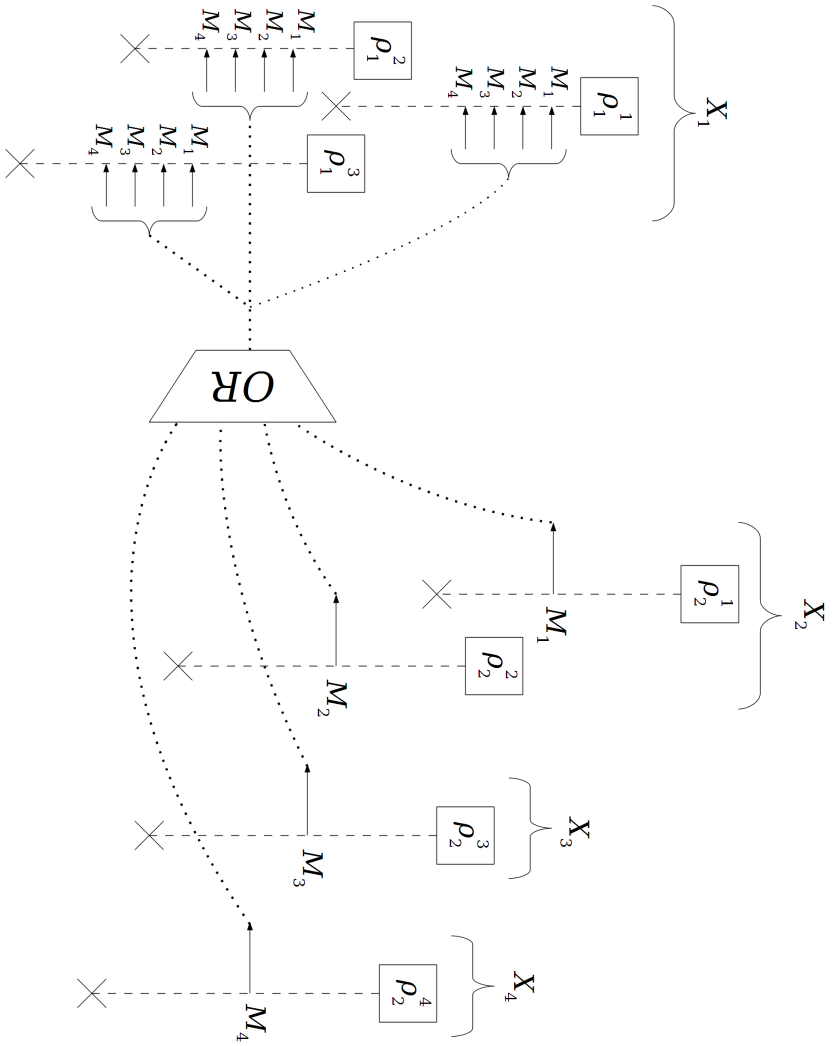


Figure 4.7: Example of canonical messages that do not constitute a binding sequence

binding sequence, i.e., a list of canonical messages assigned to M_1, M_2, M_3, M_4 in a single run of $\rho_2(.)$ is sent to a run of $\rho_1(.)$.

The receiver of a binding sequence may not know the identity of the party who has sent the list; in the figure, e.g., the run ρ_1^1 may not be able to conclude that from which run of $\rho_2(.)$ the binding sequence was sent. Similarly, the receiver of a binding sequence may not know whether the binding sequence is fresh or it is being replayed; this is indicated in the figure with the output of OR-block, i.e., the same binding sequence is accepted by multiple runs: ρ_1^1, ρ_1^2, and ρ_1^3.

A list of canonical messages may not be a binding sequence, because an adversary can mix the canonical messages from different sessions. This is illustrated in Fig. 4.7 by modifying the example of Fig. 4.6. Here, the received messages, although are canonical, are assigned in different runs of the role program $\rho_2(.)$.

Next, we present an operational definition that provides a method to obtain a guarantee that a list of message variables in a role program is associated with a binding sequence. This operational definition meets the requirement stated in Def. 4.6.

Let $\mathbf{D}_j$ be a local D-graph of a role program $\rho_j(.)$. The nodes of $\mathbf{D}_j$ represents message variables that are assigned by one of the m role programs of the protocol. We define two subgraphs of $\mathbf{D}_j$: $\vec{in}(*, \mathbf{D}_j)$ and $\vec{in}(k, \mathbf{D}_j)$.

The graph $\vec{in}(*, \mathbf{D}_j)$ is constructed on the messages variables corresponding to received messages in the role program $\rho_j(.)$; the received messages can be from any other role program. The graph $\vec{in}(k, \mathbf{D}_j)$ contains the nodes of $\mathbf{D}_j$ that correspond to the messages sent by the role program $\rho_k(.)$ and received by $\rho_j(.)$, for $k \neq j$. Clearly, the graph $\vec{in}(k, \mathbf{D}_j)$ is a subgraph of $\vec{in}(*, \mathbf{D}_j)$ and the following relation holds:

$$node(\vec{in}(*, \mathbf{D}_j)) = \bigcup_{k=1}^{m} node(\vec{in}(k, \mathbf{D}_j)) \tag{4.1}$$

Proposition 4.7 (Computing a Binding Sequence) *If*

- $\mathbf{D}_j$ *is a local D-graph in* $\rho_j(.)$
- *the lists* $leaf(\vec{in}(k, \mathbf{D}_j))$, *for* $1 \leq k \leq m$, *consist of canonical messages*

then $node(\vec{in}(*, \mathbf{D}_j))$ *is always assigned with a binding sequence of* $\rho_j(.)$, *in presence of an adversarial strategy* $\mathcal{I} \in \mathbf{I}_0$.

PROOF.

As per Def. 4.6, a binding sequence is generated in a partial session that has already occurred. Therefore, to show that a list of received messages **msg** is a binding sequence, two conditions must be satisfied. First, **msg** is a partial session, and, second, all messages in **msg** are canonical messages.

For the first condition, we know that a list of values represents a partial session if the list belongs to a subgraph of a local D-graph. Therefore, the first condition translates to verifying that $\mathbf{msg} \in \vec{in}(*, \mathbf{D}_j)$. If $\mathbf{D}_j$ is verifiable in $\rho_j(.)$ then $\vec{in}(*, \mathbf{D}_j)$ is also a verifiable D-graph. This is because the excluded messages, i.e., the sent messages in $\rho_j(.)$, can be included in the auxiliary arguments of the D-functions of $\vec{in}(*, \mathbf{D}_j)$. Since $\mathbf{D}_j$ is verifiable in $\rho_j(.)$, we conclude that $node(\vec{in}(*, \mathbf{D}_j))$ represents a partial session.
For the second condition, we proceed as follows.

All message variables in $node(\vec{in}(*, \mathbf{D}_j))$ must correspond to canonical messages. The D-graph $\vec{in}(*, \mathbf{D}_j)$ can be decomposed in to $m-1$ subgraphs: $\vec{in}(k, \mathbf{D}_j)$, for $1 \leq k \leq m$ and $k \neq j$. From the statement of the proposition, we know that the leafs of each of $\vec{in}(k, \mathbf{D}_j)$ are canonical messages, i.e., for every value of k, $leaf(\vec{in}(k, \mathbf{D}_j))$ consists of canonical messages.

From the definition of leaf nodes, we know that there is a directed path from each node of the graph $\vec{in}(k, \mathbf{D}_j)$ to some node in $leaf(\vec{in}(k, \mathbf{D}_j))$:

$$\forall M_i \in node(\vec{in}(k, \mathbf{D}_j)), \exists M_j \in leaf(\vec{in}(k, \mathbf{D}_j)) : path(M_i, M_j)$$

Each node in $\vec{in}(k, \mathbf{D}_j)$ represents a message that is sent by the k-th role program. From Proposition 4.5, we know that canonicity is propagated in the opposite direction on a path over the messages that are sent by the same role program. Thus, canonicity is propagated to all nodes of $\vec{in}(k, \mathbf{D}_j)$ starting from $leaf(\vec{in}(k, \mathbf{D}_j))$. Further, using Equation 4.1, we conclude that all nodes of $\vec{in}(*, \mathbf{D}_j)$ are canonical messages. This completes the proof. □

Essentially, a binding sequence of a role program is implied by the verification of a local D-graph of the program and the verification of canonicity of certain messages in the D-graph. A binding sequence correspond to a single local D-graph; multiple local D-graphs correspond to multiple binding sequences. This means that a role program may have a number of binding sequences, due to multiple (distinct) local D-graphs. In the example protocol of the previous chapter in § 4.4, the hash chain constitute a single local D-graph for the role program A^ρ and its nodes constitute a binding sequence for S^ρ.

In summary, the computation of a binding sequence involves two main steps. First, we write the role programs of a protocol and build local D-graphs for each role program. Second, we prove that nodes of each local D-graph are canonical messages. The lists of nodes in each local D-graph then represent the binding sequences for the corresponding role program.

At last, an overview of different graphical models is shown in Fig. 4.8. A global D-graph models a protocol from a global perspective and represents a session of the protocol. A local D-graph of a role program is a sub-graph of the corresponding global D-graph and therefore represents a partial session. Then, further down in the figure, we have two more sub-graphs of a local D-graph, corresponding to the binding sequence and canonicity requirements of a role program.

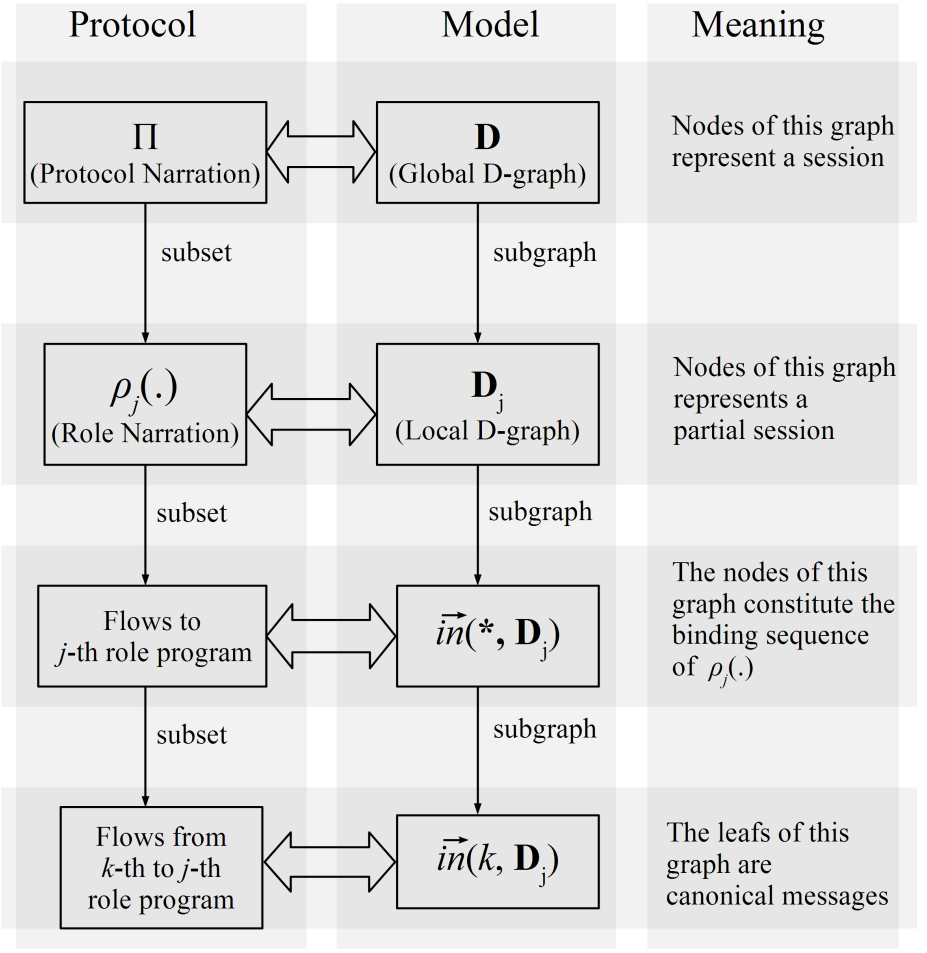


Figure 4.8: An Overview of Models and Requirements

4.4 A Simple Protocol

In this section, we present an example protocol that has a relatively simple D-graph, which brings out the basic intuition behind the notions of a D-graph and a binding sequence. We consider a two-role protocol that is based on the classic method of Lamport [93], and it is similar to, e.g., S/Key [86] and the Jane-Doe protocol [99].

The protocol uses a hash chain to generate one-time passwords for authentication. A hash chain is a sequence of values in which the next value is obtained by applying a hash function to the current value. The hash function is denoted by $\mathcal{H}(.)$. We assume that the hash function is second pre-image resistant, namely the probability of finding an input that maps to the same output, as another pre-specified input maps to, is negligible. A formal definition can be found in any standard cryptographic literature [132].

4.4.1 Protocol

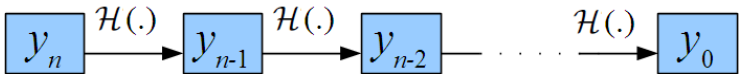
Our two-role protocol consists of two role programs S^ρ and C^ρ. The program S^ρ is executed by a server that hosts a number of client databases. The program C^ρ is executed by a client to communicate with the server in order to create, update, read, and delete its database. The server neither knows the identities of clients nor it shares any secret with the clients. The goal of the protocol is to provide an authentic channel between a database and an anonymous client.

Each database that is created on the server is protected by a password, which means that a request to add, delete, or search for any record must be accompanied by password based authentication. Let $pass$ be the password chosen by a client. The first request from a client to the server is for creating a new database. In particular, the first request contains $\mathcal{H}^n(pass)$, where $\mathcal{H}^n(.)$ stands for n successive applications of the hash function and n is a large number, e.g., $n = 1000$. The last request from a client to the server is to delete the client's database.

A message constituting the i-th request from a client to the server for accessing its database is of the form $[\mathcal{H}^{n-i}(pass), cmd_i]$, where $i < n$, cmd_i is some valid query to the database, and i can only increase monotonically in subsequent requests. A sequence of n requests constitute our protocol. Consequently, the protocol consists of n flows, and its narration is as follows.

$$
\begin{array}{ll}
(0) & C^{\rho} \longrightarrow S^{\rho} : \mathcal{H}^{n}(pass), cmd_0 \\
(1) & C^{\rho} \longrightarrow S^{\rho} : \mathcal{H}^{n-1}(pass), cmd_1 \\
(2) & C^{\rho} \longrightarrow S^{\rho} : \mathcal{H}^{n-2}(pass), cmd_2 \\
 & \cdot \\
 & \cdot \\
(n) & C^{\rho} \longrightarrow S^{\rho} : pass, cmd_n
\end{array}
$$

The protocol messages constitute a hash chain, $y_n \rightarrow y_{n-1} \rightarrow \cdots \rightarrow y_0$, where $y_i = \mathcal{H}^{n-i}(pass)$, for instance $y_n = pass$, and $y_0 = \mathcal{H}^{n}(pass)$. This hash chain is the D-graph of the protocol. In this D-graph, the direction of arrows follows the direction in which the hash function is applied, as shown below.



One can imagine that the server program also replies back to the client program, but such replies are of no interest to us. Similarly, a client may choose to terminate the protocol before making n requests, however, such cases can be included by assuming n to be a run-time variable rather than a constant.

The protocol is executed in a network that connects a large number of clients to a server. Each client can execute multiple copies of the role program C^{ρ} using different passwords in order to manage multiple databases. The server runs multiple copies of S^{ρ} in parallel to serve all active runs of the client program.

We assume the presence of a weak network adversary:

- The adversary is computationally bounded.[6]
- The adversary is an insider, i.e., the adversary can act as a client.
- The adversary can write a new message, read a messages of another client, and delete a messages sent by another client.
- The adversary cannot modify a message sent by another client, i.e., he cannot delete and read a message at the same time.

This last assumption is reasonable in a wireless broadcast scenario, where it is hard for an adversary to jam and read messages at the same time [130]. If an adversary can delete and read a message at the same time then clearly the adversary can change the client's request to his will.

[6]This assumption is implied by the second pre-image resistance of a hash function.

An adversary can also invoke a number of runs of C^ρ to create his own databases on the server. We, however, assume that an adversary cannot act as a server, which is a reasonable assumption if the routing address of the server is known. Inevitably, there are many tacit assumptions regarding maximum size of a database, a limit on the life time of a database, maximum size of the data embedded in a request, randomness of passwords, etc.

4.4.2 Properties

When S^ρ receives y_n in the last flow, the computation of the hash chain is complete in S^ρ. We are interested in the information that the server obtains after a DC-run of S^ρ. Some of the information is obvious, such as the server knows which requests were made and whether a hash chain was computed correctly by a client. The server's ability to verify a hash chain (D-graph) is important in our methodology, but here we choose to highlight another aspect of the protocol, which is as follows.

After a DC-run of S^ρ, the server concludes that, with a high probability, there must be a matching run of C^ρ that has previously occurred on a client. In other words, there is just one copy of the client program (with a single password) with whom a copy of the server program has communicated. In the following, we analyse why the D-graph of the protocol, i.e., the hash chain, implies a matching run.

The underlying network provides an atomic channel for the duration of a single request, because an adversary cannot modify a request. The first request to create a new database is a (pseudo) authentic message[7] sent by an anonymous client; the message is authentic because there is no database associated with the client initially. Thus, the authenticity of the first request is guaranteed by the setup assumptions.

After the first request a new database is crated for the client. Now, subsequent requests to manage the newly crated database are considered authentic only if they are from the same client. The authenticity of subsequent requests cannot be guaranteed by only using the setup assumptions, because an adversary may able to send subsequent requests on behalf of a client.

The following observation is important: a guarantee of authenticity propagates on a hash chain, from a node to the previous node (the reverse direction). This

[7]The identity of the party who sends a pseudo authentic message is not important [113]. For brevity, we are using the term authenticity to mean pseudo authenticity in this example.

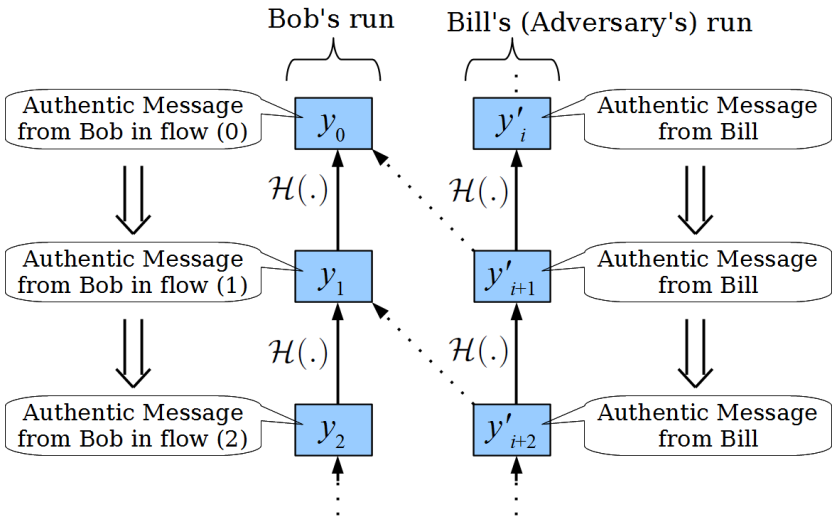


Figure 4.9: Prorogation of authenticity in reverse direction

means that if the value of y_0 is an authentic message then so does the values of $y_1, \ldots, y_n$. Unless an adversary finds a second pre-image in the hash function, it cannot be the case that the client program has generated the values in any of its n flows that differ from the values that the server program has received.

The propagation of authenticity is illustrated in Fig. 4.9, where two hash chains corresponding to two runs of the client program C^ρ are shown. The hash chain in the left side of the figure is generated when a client Bob executes C^ρ, and the hash chain on the right side is generated when a client Bill (who may be an adversary) executes C^ρ. The arrow from y'_{i+2} to y_1 represents the case if the two runs of C^ρ generate two different values for y_2 and y'_{i+2} such that $\mathcal{H}(y_2) = \mathcal{H}(y'_{i+2})$, where $0 \leq i \leq n-2$. This means that the adversary has successfully found two values, y_2 and y'_{i+2}, that generate a collision in the hash function $\mathcal{H}(.)$. Since we assume that $\mathcal{H}(.)$ is second pre-image resistant, it is reasonable to conclude that an authentic message y_1, in flow (1), implies an authentic message y_2, in flow (2).

If the client's password remains secret then the values for $y_1, \ldots, y_n$ are all sent by the same client running C^ρ. Therefore, a DC-run of S^ρ implies a matching run of C^ρ executed by the client who knows *pass*.

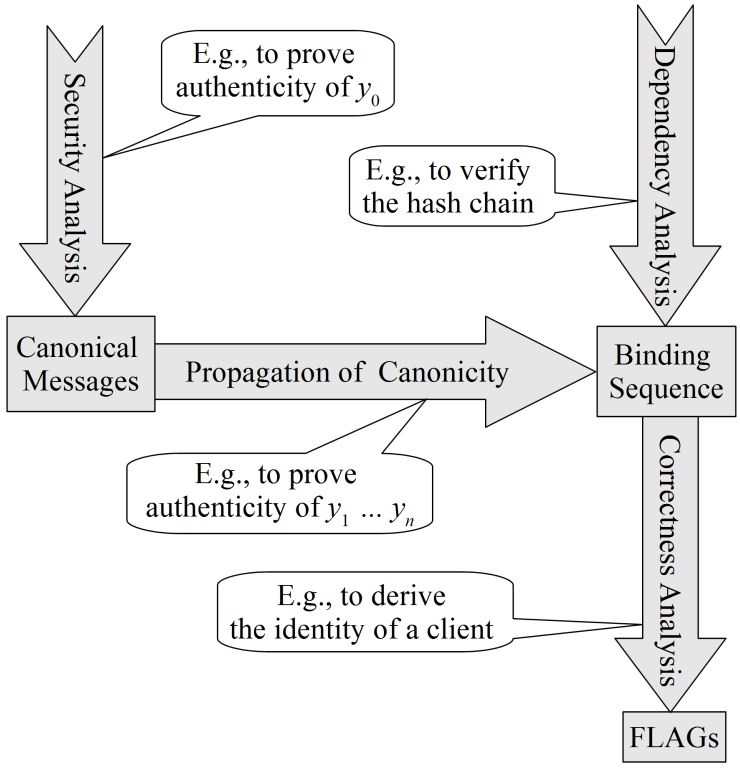


Figure 4.10: Overview of the analysis of the example protocol

4.4.3 Binding Sequence

For our two-role protocol, a list consisting of authentic messages that is sent in a single run of the client role program is a *binding sequence* of the server role program. In fact, we do not require messages to be authentic; we only require the messages to be canonical, which is a weak form of message authentication, as discussed before.

A binding sequence, which consists of canonical messages, is sent by some honest party. The receiver of a binding sequence may not know the identity of the sender. Similarly, a binding sequence may not provide any guarantee for its freshness, i.e., a binding sequence can be replayed by an adversary.

A binding sequence is sufficient to derive fine level authentication goals (FLAGs) without relying on any other security property. Authentication goals are derived

from a binding sequence in a constructive manner, e.g., we say that the server executing S^ρ achieves the identification of a client executing C^ρ if the program S^ρ can locally compute a function that maps S^ρ's binding sequence to the the identity of the client. The construction of such a function in S^ρ serves as a proof that the role program S^ρ achieves the identification goal for a client.
The overall structure of our analysis is shown in Fig. 4.10.

Coming back to our example, the reader may have noticed that the trivial verifiability of the D-graph (hash chain) in S^ρ is due to our choice of this simple protocol. In general, a party may not know all of the values of the terms occurring in a protocol. Similarly, the proof of canonicity (or authenticity) for a received message is a non-trivial task, because an adversary may be able to insert new messages. These issues are discussed in the next chapter. In this chapter, we only introduce the concept of a dependency graph, without deliberating its verifiability aspect.

4.5 Case Study 1

We once again consider the five-pass authentication protocol from ISO-9798 part 2, which is listed below:

(1) $A^\rho \longrightarrow B^\rho$: $M_1 = R_A$
(2) $B^\rho \longrightarrow S^\rho$: $M_2 = R'_B, M_3 = R_A, A$
(3) $S^\rho \longrightarrow B^\rho$: $M_4 = \mathcal{E}_{K_{BS}}(R'_B \| K_{AB} \| A), M_5 = \mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B)$
(4) $B^\rho \longrightarrow A^\rho$: $M_6 = \mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B), M_7 = \mathcal{E}_{K_{AB}}(R_B \| R_A)$
(5) $A^\rho \longrightarrow B^\rho$: $M_8 = \mathcal{E}_{K_{AB}}(R_A \| R_B)$

The protocol consists of three role programs. The narration of a role program can be obtained from the protocol narration by omitting those flows in which the role program is not a communication partner. The analysis for the three role programs is in the following.

4.5.1 Initiator program: $A^\rho \stackrel{\text{def}}{=} \rho_1(\bar{R}_A, \bar{K}_{AS}, \bar{A}, \bar{B})$

The role narration is as follows.

(1) $A^\rho \rightarrow B^\rho$: $M_1 = R_A$
(4) $B^\rho \rightarrow A^\rho$: $M_6 = \mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B), M_7 = \mathcal{E}_{K_{AB}}(R_B \| R_A)$
(5) $A^\rho \rightarrow B^\rho$: $M_8 = \mathcal{E}_{K_{AB}}(R_A \| R_B)$

The narration of A^ρ consists of four messages variables. The four messages constitute a single local D-graph, which we denote by $\mathbf{D}_1$ and is shown in Fig. 4.11-(a). The local D-graph contains four arcs corresponding to four D-functions. These D-functions are clear from the role narration and are listed below.

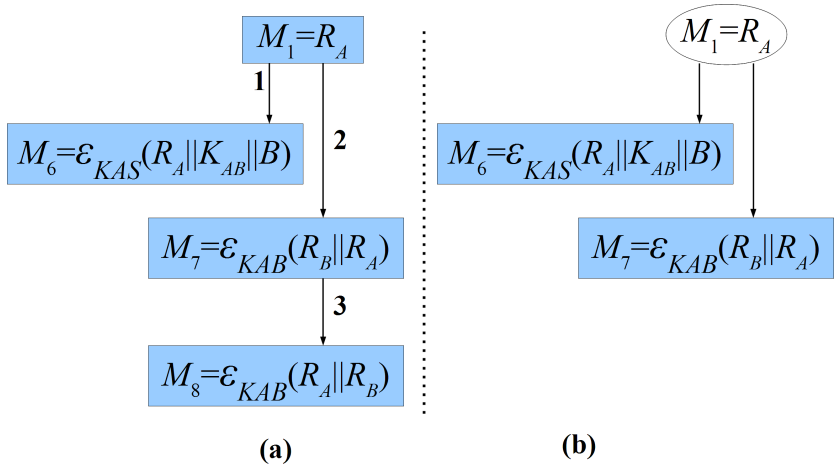


Figure 4.11: (a) Local D-graph of A^ρ (b) D-graph on received messages

(1) $M_6 = Dep_1(M_1, K_{AS}, K_{AB}, B)$**:** $return(\mathcal{E}_{K_{AS}}(M_1 \| K_{AB} \| B))$

(2) $M_7 = Dep_2(M_1, K_{AB}, R_B)$**:** $return(\mathcal{E}_{K_{AB}}(R_B \| M_1))$

(3) $M_8 = Dep_1(M_7, K_{AB})$**:** $x_1, x_2 \leftarrow \mathcal{D}_{K_{AB}}(M_7)$;
$return(\mathcal{E}_{K_{AB}}(x_2 \| x_1))$

Since A^ρ only communicates with the role program B^ρ, the two subgraphs $\vec{in}(2, \mathbf{D}_1)$ and $\vec{in}(*, \mathbf{D}_1)$ are the same. The graph $\vec{in}(2, \mathbf{D}_1)$ is shown in Fig. 4.11-(b), in which there are two leaf nodes, M_6 and M_7. These two variables are required to be assigned with canonical messages. If the canonicity can be verified then the binding sequence is $\beta_1 = node(\vec{in}(*, \mathbf{D}_1)) = [M_6, M_7]$. We later describe the canonicity verification using a model checker.

4.5.2 Responder program: $B^\rho \stackrel{\text{def}}{=} \rho_2(\bar{R}_B, \bar{R}'_B, \bar{K}_{BS}, \bar{A}, \bar{B})$

The narration of the role program B^ρ is the same as the protocol narration, because B^ρ takes part in all the flows. The role narration is as follows.

(1) $A^\rho \rightarrow B^\rho$: $M_1 = R_A$
(2) $B^\rho \rightarrow S^\rho$: $M_2 = R'_B, M_3 = R_A, A$
(3) $S^\rho \rightarrow B^\rho$: $M_4 = \mathcal{E}_{K_{BS}}(R'_B \| K_{AB} \| A), M_5 = \mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B)$
(4) $B^\rho \rightarrow A^\rho$: $M_6 = \mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B), M_7 = \mathcal{E}_{K_{AB}}(R_B \| R_A)$
(5) $A^\rho \rightarrow B^\rho$: $M_8 = \mathcal{E}_{K_{AB}}(R_A \| R_B)$

The local D-graphs constructed on the message variables of B^ρ is shown in Fig. 4.12-(a). Note that the message variable M_5 is not included as a node, because this message cannot be decrypted by B^ρ and the dependency function linking M_5 to the D-graph cannot be verified by B^ρ. The underlying D-functions can easily be constructed, as listed below.

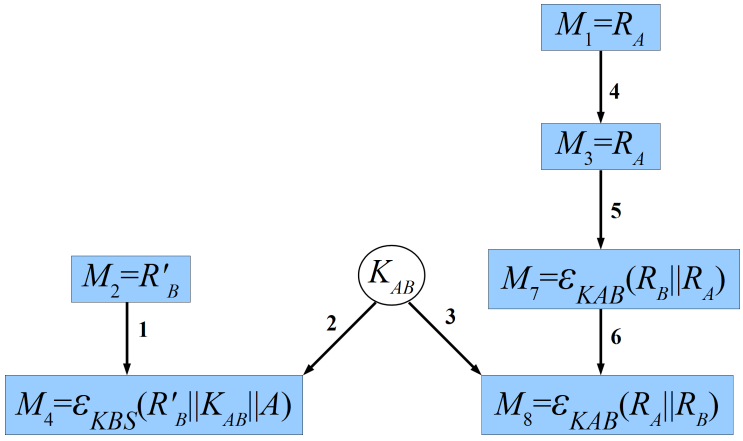


Figure 4.12: Local Dependency Graph of B^ρ, $\mathbf{D}_2$

(1) $M_4 = Dep_1(M_2, K_{BS}, K_{AB}, A)$**:** $return(\mathcal{E}_{K_{BS}}(M_2 \| K_{AB} \| A))$

(2) $M_4 = Dep_2(K_{AB}, K_{BS}, R'_B, A)$**:** $return(\mathcal{E}_{K_{BS}}(R'_B \| K_{AB} \| A))$

(3) $M_8 = Dep_3(K_{AB}, R_A, R_B)$**:** $return(\mathcal{E}_{K_{AB}}(R_A \| R_B))$

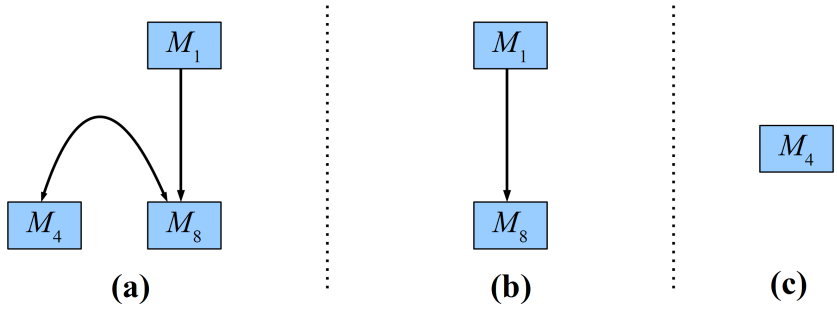


Figure 4.13: (a) The graph $\vec{in}(*, \mathbf{D}_2)$ (b) $\vec{in}(1, in(\mathbf{D}_2))$ (c) $\vec{in}(3, in(\mathbf{D}_2))$

(4) $M_3 = Dep_4(M_1)$**:** $return(M_3)$

(5) $M_7 = Dep_5(M_3, K_{AB}, R_B)$**:** $return(\mathcal{E}_{K_{AB}}(R_B \| M_3))$

(6) $M_8 = Dep_6(M_7, K_{AB})$**:** $x_1, x_2 \leftarrow \mathcal{D}_{K_{AB}}(M_7)$;
$return(\mathcal{E}_{K_{AB}}(x_2 \| x_1))$

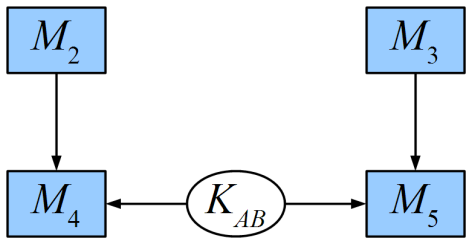
The graphs $\vec{in}(*, \mathbf{D}_2)$, $\vec{in}(1, \mathbf{D}_2)$, and $\vec{in}(3, \mathbf{D}_2)$ are shown in Fig. 4.13. The message variables M_4 and M_8 are the leafs in $\vec{in}(1, \mathbf{D}_2)$ and $\vec{in}(3, \mathbf{D}_2)$, therefore, we need to prove the canonicity of messages assigned to M_4 and M_8. If the canaonicity requirement can be satisfied then the binding sequence is $\beta_2 = leaf(\vec{in}(*, \mathbf{D}_2)) = [M_1, M_4, M_8]$

4.5.3 TTP program: $S^\rho \stackrel{\text{def}}{=} \rho_3(\bar{K}_{AS}, \bar{K}_{BS}, \bar{K}_{AB}, \bar{A}, \bar{B})$

A trusted third party (TTP) S executes the role program S^ρ. The trusted party shares long term secrets with all network parties. The narration of the role program S^ρ is as follows.

(2) $B^\rho \rightarrow S^\rho$: $M_2 = R'_B, M_3 = R_A, M'_3 = A$
(3) $S^\rho \rightarrow B^\rho$: $M_4 = \mathcal{E}_{K_{BS}}(R'_B \| K_{AB} \| A), M_5 = \mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B)$

The local D-graph of S^ρ is shown below.



In the D-graph, for instance, $K_{AB} \rightarrow M_4$ is a D-arc that corresponds to the D-function $M_4 = Dep(K_{AB}, K_{BS}, R'_B, A)$. In this D-graph, there are two received message variables, M_2 and M_3. We do no further analysis, because received messages for M_2 and M_3 are plaintext messages and cannot be the candidates of canonical messages against a realistic adversary.

4.5.4 Canonicity Analysis

The canonicity of M_4, M_6, M_7, and M_8 can be analysed with different methods and using different attacker model. For instance, in a cryptographic model, the canonicity of $M_6 = \mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B)$, which is a message sent by the server program, can be proved by the following reduction type argument:
If an adversary $\mathcal{I} \in \mathbf{I}_0$ can compute M_6 on a value of R_A of his choice, without using the server program, then $\mathcal{I}$ can be used to break the encryption scheme, assuming that K_{AS} is only shared between A and S, and that A and S are honest parties.

In this case study, we rely on an abstract analysis. We use a symbolic model checker, called OFMC (open-source fixed-point model checker), which uses the classic Dolev-Yao attack model [62] and supports an insider adversary. An insider adversary is a legitimate network user. In our case, this means that an adversary shares a long term secret key with the trusted third party S and can executes the role programs A^ρ and B^ρ. This attacker model is an abstraction of our adversary class $\mathbf{I}_0$.

We analyse the protocol with the security goal of non-injective agreement on M_4, M_6, M_7, and M_8. A non-injective agreement on a message means that the message is a canonical message and is sent by a particular party. Therefore, the non-injective agreement is a stronger security property than canonicity. The analysis results show that the protocol achieves a non-injective agreement on M_4, M_6, M_7, and M_8, which means that these message variables correspond to canonical messages. Further details of the verification are in Appendix A.

This completes the security analysis of our protocol with the following results:

- The binding sequence of A^ρ is $\beta_1 = [M_6, M_7]$.
- The binding sequence of B^ρ is $\beta_2 = [M_1, M_4, M_8]$

The tool OFMC, along with the tutorial and examples, is available online:
`http://www2.imm.dtu.dk/~samo`

4.6 Summary

In this chapter, we presented an execution model of a protocol in terms of role programs. We introduced the notion of a local D-graph, which is constructed on the messages of a role program, using only the local perspective of the role program. It is required that the D-functions of a local D-graph are verifiable in the role program. An instance of a local D-graph is a partial session, since a role program does not participate in every flow of the protocol. We introduced the notion of a canonical message (which is orthogonal to the notions of a D-function and a D-graph). If a receiver of a message has the guarantee that this message is not sent by an adversary and it is sent by an expected role program then the message is called a canonical message. The receiver of a canonical message may not know the identity of the sender. We also introduced the notion of a binding sequence, which is a list of canonical messages that are linked together by a local D-graph. This means that a binding sequence is a list of values sent by some honest party in a single partial session. At last, we analysed the protocol of our case study to demonstrate how to derive its binding sequences.

Chapter 5

Authentication Goals

In this chapter, we present the third and last step of the SI methodology, which is related to the derivation of fine level authentication goals (FLAGs). FLAGs are the correctness requirements of an entity authentication protocol. Therefore, the derivation process of FLAGs is also called the correctness analysis. Two authentication protocols are different for an application (which uses authentication as a service) if their sets of FLAGs are different. Of course, to achieve a certain FLAG, different protocols employ different cryptographic techniques and are based on different setup assumptions, e.g., nonces vs. time-stamps and public-key vs. symmetric-key ciphers.

In the SI methodology, a FLAG of a role program is derived from a binding sequences of the role program. (A role program may have multiple binding sequences.) This derivation process does not consider the dynamic behaviour of the role program, which can be affected by a network adversary. This is because a binding sequence is used as an assumption in the correctness analysis. This allows us to ignore the role of a network adversary. Therefore, we consider the correctness analysis as a non-security analysis.

With respect to a role program, the validity of a FLAG depends on an adversarial model, because the binding sequence used to derive the FLAG is a result of a security analysis, which explicitly considers the dynamic behaviour of the protocol in presence of a network adversary. The notion of a binding sequence

allows us to demarcate the correctness analysis (the derivation of FLAGs from a binding sequence) from the security analysis (the derivation of the binding sequence).

We define each FLAG from two perspectives: a conceptual perspective and an operational perspective. A conceptual definition describes the concept behind a FLAG in a natural language and without being protocol specific. A conceptual definition considers a FLAG as a service as seen from an application point of view. The operational definition of a FLAG describes the FLAG for a security analyst so that he can verify the FLAG, namely an operational definition is a concrete procedure that decides whether or not a FLAG is achieved by a role program.

In Chapter 1, we have already introduced the conceptual definitions of FLAGs. In the next section, we simply rephrase these definitions in the context of various notions that we introduced since Chapter 1.

5.1 Conceptual Definitions

We start by explaining the importance of conceptual definitions. A conceptual definition specifies the service aspect of a FLAG, which is important to a system designer, who may not be a security expert and who mainly uses an authentication protocol as a service in a design process. In our view, the question why a protocol achieves a particular FLAG is not important in the design of a larger system. On the other hand, a system designer may well be interested in resource requirements and setup assumptions accompanied by an authentication protocol. The specification of such aspects, however, can be quite independent of why protocol works and achieves a set of FLAGs.

We define FLAGs from the local perspective of a role program. In the following, we use the variable A to represent the the local party for which a FLAG is being defined, and we use B and C to represent two other network parties, where $A \neq B \neq C$. The party A is the verifier and the party B is the claimant in the authentication process.

Definition 5.1 (FLAGs) The conceptual definitions of FLAGs are as follows:

(1) Recognition: If A verifies that B is the same entity with whom A has communicated before then A is said to achieve the *recognition* of B, denoted by the predicate $\mathit{Recog}(A \rhd B, .)$.

(2) Identification: Let *db* be an identification database that is accessible to A and trusted by A. If A verifies that the claimed identity of B can be linked to a specific record of *db* then A is said to achieve the identification of B, denoted by the predicate $Idnt(A \triangleright B, .)$.

(3) Operativeness: If A verifies that the party who claims to be B is currently active on the network then A is said to achieve the operativeness of B, denoted by the predicate $Oper(A \triangleright B, .)$.

(4) Willingness If an entity A verifies that the party who claims to be B once wanted to communicate to A then A is said to achieve the willingness of B, denoted by the predicate $Wlng(A \triangleright B, .)$.

(5) One-sided Authentication If an entity A verifies that an identified peer entity B is currently ready to communicate with A then one-sided authentication is achieved, denoted by the predicate $\text{OATH}(A \triangleright B, .)$:

$$\text{OATH}(A \triangleright B, .) = Idnt(A \triangleright B, .) \wedge Wlng(A \triangleright B, .) \wedge Oper(A \triangleright B, .)$$

Here, the use of *logical and* $\wedge$ implies that the goals $Idnt(A \triangleright B, .)$, $Wlng(A \triangleright B, .)$, and $Oper(A \triangleright B, .)$ are all achieved together. As it will become clear in the next section, achieving two FLAGs together actually means that both FLAGs are derived from the same binding sequence.

(6) Pseudo One-sided Authentication If an entity A verifies that a recognized peer entity B is currently ready to communicate with A then pseudo one-sided authentication is achieved, denoted by the predicate $\text{Pseudo-OATH}(A \triangleright B, .)$:

$$\text{Pseudo-OATH}(A \triangleright B, .) = Recog(A \triangleright B, .) \wedge Wlng(A \triangleright B, .) \wedge Oper(A \triangleright B, .)$$

(7) Confirmation If an entity A verifies that the peer entity B knows that a FLAG G has been achieved then A is said to achieve a confirmation on G from B, denoted by the predicate $Cnfm(A \triangleright B, G, .)$.

(8) Strong One-sided Authentication If an entity A verifies that an identified peer entity B is currently ready to communicate with A and B knows this belief of A then strong one-sided authentication is achieved, denoted by the predicate $\text{Str-OATH}(A \triangleright B, .)$:

$$\text{Str-OATH}(A \triangleright B, .) = \text{OATH}(A \triangleright B, .) \wedge Cnfm\big(A \triangleright B, \text{OATH}(A \triangleright B, .), .\big)$$

(9) Mutual Authentication If an entity A verifies that A and the peer entity B currently want to communicate with each other, then A is said to achieve mutual authentication, denoted by $\text{MATH}(A \triangleright B, .)$:

$$\text{MATH}(A \triangleright B, .) = \text{OATH}(A \triangleright B, .) \wedge Cnfm\big(A \triangleright B, \text{OATH}(B \triangleright A, .), .\big)$$

A few remarks about each of the above definitions are in order.
The goal recognition does not require that A already knows B. For entity recognition, the real identity of B is not important. Entity recognition only makes sense for two or more instances of interaction between A and B. As the definition specifies, entity recognition is achieved if in a later instance of communication A can verify that B is the same party with whom it has communicated before.

The second FLAG is entity identification, which requires that A is able to verify the claimed identity of B. Entity identification assumes that A already knows the identities of all network entities in form of an identification database. The identification database is a kind of abstraction and it does not imply that A actually stores the database locally. In the definition, we only assume that A has an access to the identification database, e.g., B may present its claimed identity to A signed by a trusted party in a common public key infrastructure (PKI). In this way, using PKI, A can compute whether the claimed identity is a valid network identity.

The third FLAG is the operativeness of a peer entity, which requires that the peer entity B is currently active on the network. This goal does not require that A verifies the identity of B. The meaning of currently active can be interpreted in slightly different manners depending on the type of nonce used by A, which we formalize in the corresponding operational definition. For instance, if the nonce of A is a time-stamp then the period of the time stamp determines the meaning of currently active; if the time period is 5-seconds then the operativeness of B implies that B was there within the 5-second window. On the other hand, if a sufficiently large random number is used then the operativeness means that the party pretending to be B has actually participated in the current execution of the protocol.

The fourth FLAG is the willingness of a party, which requires that the party pretending to be B provides its consent to communicate with A. Clearly, this requires that somehow B can identify A and messages sent by B to A are specifically generated for A. The willingness goal can be used to avoid many relay and man-in-middle attacks.

The fifth FLAG is one-sided authentication, which is achieved if A achieves the identification, operativeness, and willingness of B together. (As we see later, *together* means achieving these FLAG from the same binding sequence.) Achieving these three goals ensures three things: the party who claims to be B is indeed B; B is currently there; and B wants to communicate with A. In other words, B is currently ready to communicate with A. The next FLAG, pseudo one-sided authentication, is the same as one-sided authentication, except the identification requirement is replaced by the recognition requirement.

It is important to note that for one-sided authentication the three FLAGs must be achieved together; e.g., if the three FLAGs are achieved using three different protocols then one cannot conclude that one-sided authentication is achieved. This is because willingness and operativeness do not require the validation of the claimed identity of B, and therefore the role of B can be played by different parties.

The seventh FLAG is the confirmation of another FLAG. In a sense, the confirmation is a higher order goal. To understand its role, note that each FLAG is defined from the perspective of A, therefore, e.g., if a far-end party B authenticates A then A may not know about this authentication. For A, one way of determining the occurrence of this authentication event is to ask a confirmation message from B.

The eighth FLAG is strong one-sided authentication, which means that not only does A achieve one-sided authentication of B but A also confirms that B knows that A has achieved this goal. This type of assurance is typically required in applications where a subsequent action is expected from parties, e.g., without any confirmation from B, A may start streaming a TV channel to B while B is re-authenticating itself to A.

The last goal in the list is mutual authentication, which requires that both A and B achieve one-sided authentication. Since FLAGs are defined from the local perspective of A, the one-sided authentication of A by B is conveyed to A by the confirmation goal. Note that merely achieving one-sided authentication twice does not imply mutual authentication; the two one-sided authentication must be achieved together, from the same binding sequence.

For instance, if A and B use a one-sided authentication protocol and execute two copies of the protocol in a reciprocal manner then the mutual authentication is not achieved as per our definition. In many applications, undesirable behaviours occur if there is no distinction between two one-sided authentication and mutual authentication. Consider a situation in which A and B execute multiple sessions with each other in parallel. Without any link between the two one-sided authentication events, the two events may occur in two different sessions. Similarly, if an authentication protocol establishes a session key besides entity authentication then it is important that both parties authenticate each other in the same session and with respect to the same value of key.

A new FLAG can be defined by combining different FLAGs, e.g., a combination of willingness and recognition is a new FLAG. Similarly, the goal confirmation can be applied to any other goal, e.g., the confirmation on mutual authentication implies strong mutual authentication. In this way, the above list of FLAGs can be extended, but, even with such extension, we do not claim that our list of

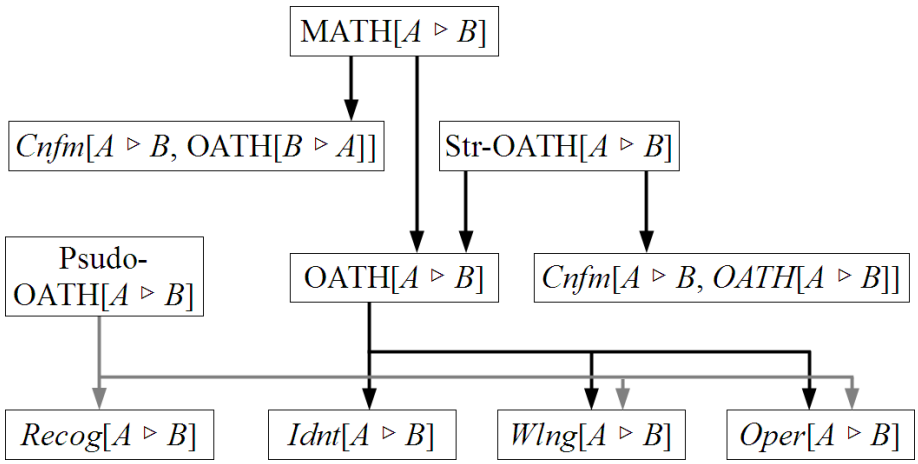


Figure 5.1: Partial order between FLAGs

FLAGs is complete The presented list is merely based on our experience and represents commonly expected authentication goals.

It is also clear from Def. 5.1 that there is a partial order between FLAGs. The hierarchical relations between FLAGs that are valid (by definition) are shown in Fig. 5.1. The FLAGs that appear as the leafs of the graph are called low-level FLAGs, and the rest of the FLAGs are called high-level FLAGs. A high-level FLAG depends on low-level FLAGs.

The conceptual definitions of FLAGs specify the service aspect of entity authentication, which is independent of a security model or a concrete protocol. To show that a FLAG is achieved by an authentication protocol, a security analyst needs operational definitions of FLAGs, which are presented in the next section.

5.2 Operational Definitions

Formulating the operational definitions of FLAGs require a bit of extra care, because, first, the link between a conceptual definition and an operational definition is somewhat intuitive. Second, if an operational definition is overly specific to a protocol structure and security model then it becomes hard to generalise. As stated earlier, the abstraction that we use in our operational definitions is that of a binding sequence, which allows the application of our operational

definitions to a wide variety of authentication protocols. In particular, if two different role programs have the same input and have the same binding sequence then the both programs achieve the same set of FLAGs.

As described in the previous chapter, a binding sequence is a list of messages that is generated in one session. This means that if a role program accepts a sequence of messages as a binding sequence then the program can assume certain restrictions on the behaviour of a network adversary. In particular, an adversary cannot modify a message in a binding sequence or mix different binding sequences, because this violates the very definition of a binding sequence. If an adversary can modify a message then the message cannot be a canonical message and therefore the message cannot be a part of a binding sequence.

An adversary, however, can present a receiver with any valid binding sequence, because a binding sequence is neither linked to a particular sender party nor does a binding sequence imply freshness of the messages. The main challenges for the receiver of a binding sequence are, therefore, to decide whether the binding sequence is from the party who claimed to be its sender and whether the binding sequence is being replayed. These and similar challenges are the crux of the correctness analysis and their requirements are embodied in the operational definition of FLAGs.

We define the operational definitions of FLAGs in a complexity theoretic framework using asymptotic notions. We believe that from these operational procedures deriving the corresponding operational procedures for other formalisms is not too hard. The operational definitions are based on a type of algorithms, which we call the *distinguisher* and which decide whether or not a challenge sent by an adversary is computed in a particular network configuration.

Definition 5.2 (Distinguisher) Let $\mathbf{C}_1$ and $\mathbf{C}_0$ be the two authentication challenges that correspond to two different network configurations. One of the challenges C_b is presented to a role program $\rho_j(\overline{\mathbf{const}}_j)$, where the value of b is selected by an adversary. A distinguisher $dst(C_b, \overline{\mathbf{const}}_j[i])$ in $\rho_j(\overline{\mathbf{const}}_j)$ is an efficient algorithm that determines the correct value of b with a high probability p_h, where $p_h = 1 - \varepsilon$, for a negligible ε.

The value of b is assigned by an adversary, therefore we cannot always associate a probability distribution to b. In some protocols, an adversary cannot freely assign a value to b, because one of the challenges can be inconsistent with the security assumptions. For example, it may be the case that if an adversary chooses $\mathbf{C}_0$ as an authentication challenge then this implies that the adversary can compute the private key of a party; since the confidentiality of private keys is a standard security assumption, one concludes that $\mathbf{C}_0$ cannot occur. In such

cases, there is no need to construct a distinguisher.

All high-level FLAGs are defined in terms of low-level FLAGs. For a low-level FLAG, we specify two network configurations that generate the two authentication challenges of the FLAG. One of the configuration corresponds to the case when the FLAG is achieved. This is called the expected configuration of the FLAG and is denoted by $\mathbf{C}_1$. The other configuration represents the alternate configuration, namely a configuration in which the FLAG is not achieved, and it is denoted by $\mathbf{C}_0$.

A network adversary can select any of the above two challenges (the value of b) and present the selected challenge to a role program. The role program does not have an a priori knowledge about the adversary's selection. Now, if a security analyst is able to construct a distinguisher within the role program that correctly determines the value of b with a high probability then the construction of the distinguisher serves as a proof that the FLAG is achieved by the role program.

If a role program determines, with the help of a distinguisher implemented in it, that the value of b is 1 then the program concludes that the challenge is received from the expected configuration. On the other hand, if $b = 0$ then the role program concludes that the challenge is from an alternate configuration.

If a role program is designed to achieve a FLAG G then a run of the role program must only succeed if the distinguisher of G generates 1. Typically, a role program achieves a number of FLAGs, and each FLAG correspond to an implementation of the corresponding distinguisher. A distinguisher is not an additional verification algorithm at the end of a role program, as it may seem from the above description. In fact, the functions inside a distinguisher are a subset of the dependency functions of a role program. Therefore, when a role program verifies its dependency graphs, all of the distinguishers of the role program are also computed. The verification of local dependency graphs fails if any of the distinguishers generates 0. That is why, we do not include distinguisher algorithms in the following definition of a successful run.

Definition 5.3 (Successful Run) A dependency compliant (DC) run, as per Def. 4.3, is a successful run.

The relations between different types of verification functions are illustrated in Fig. 5.2-(a), which shows that dependency functions encompass semantic checks as well as distinguisher functions. From a methodological point of view, it is, however, essential to make distinctions among these three types of functions, as illustrated in Fig. 5.2-(b). The semantic checks are performed during the communication phase of a role program. Although the semantic checks are not

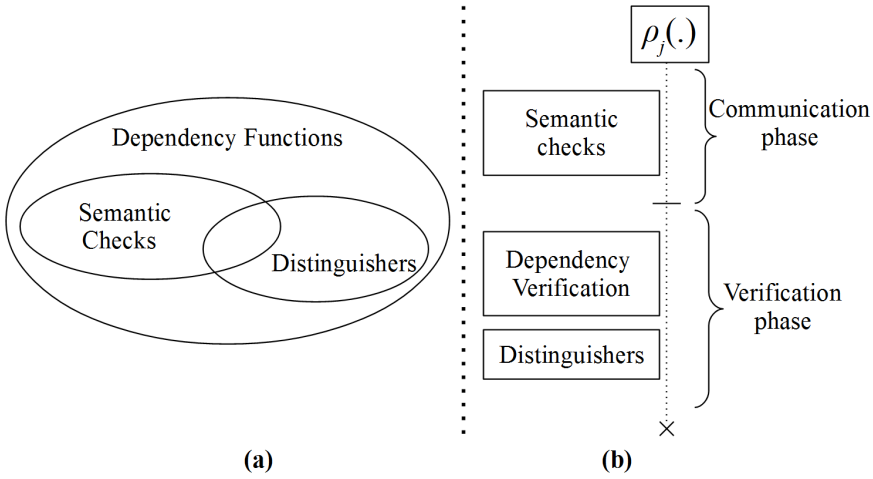


Figure 5.2: Distinguisher, Semantic checks, and Dependency Functions

used in the SI methodology, we recognize their importance in avoiding trivial denial of service attacks. The distinguishers of a role program are used to assert that certain FLAGs are achieved, and for the analysis purpose we consider them as a separate set of algorithms at the end of a role program. Since distinguisher functions constitute a subset of the dependency functions of a role program, the distinguisher functions do not need to be implemented; if the dependency functions are verified then the distinguisher functions are also verified.

As described previously, we use a *bar* on a term if the term is a constant. A constant is a value that is passed to a role program by the calling routine, and it is the value that is known at the start of a run. On the other hand, a variable is assigned a value during a run; the value assigned to a variable is a function of received messages and constants.

5.2.1 Recognition

The recognition goal does not depend on the identity of the local party that executes the role program. Therefore, the recognition of a far-end party is defined for a role program, instead of a party that executes the role program. For a straight forward comparison to the conceptual definition, we assume that the role program that achieves the recognition of B is executed by a party A. Since this assumption is not necessary, A can be replaced with A^{ρ} in the

following definition of recognition.

Definition 5.4 (Recognition) Let

1. $\beta_j(r)$, $\beta_j(r')$ and $\beta_j(r'')$ be the three binding sequences that are received in three different runs of a role program $\rho_j(\overline{\mathbf{const}}_j)$.
2. another role program $\rho_{j'}(\overline{\mathbf{const}}_{j'})$ sends these three binding sequences.
3. the role program $\rho_j(\overline{\mathbf{const}}_j)$ is executed by a party A in the three sessions.
4. the role program $\rho_{j'}(\overline{\mathbf{const}}_{j'})$ is executed by B in the first and the third session, and is executed by C in the second session, where $C \neq B$.
5. the two challenges be $\mathbf{C}_0 = (\beta_j(r), \beta_j(r'))$ and $\mathbf{C}_1 = (\beta_j(r), \beta_j(r''))$.

The party A is said to achieve $Recog(A \triangleright B, \beta_A)$ (the recognition of B by A based on β_j) if one of the following conditions holds.

- The challenge $\mathbf{C}_0$ is inconsistent with security assumptions.
- A distinguisher $dst^{rcog}(C_b, \overline{\mathbf{const}}_j)$ can be constructed in $\rho_j(\overline{\mathbf{const}}_j)$ for all choices of B and C.

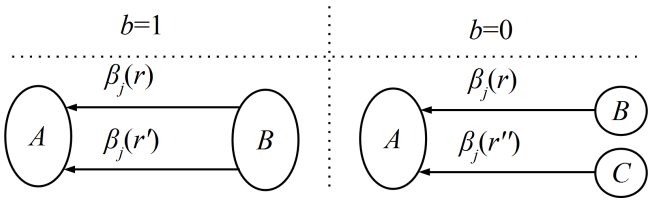


Figure 5.3: Configurations for the Distinguishers of $Recog(A \triangleright B, .)$

This definition is also illustrated in Fig. 5.3. In the definition, the challenge $\mathbf{C}_1$ represents the situation when the recognition of B is achieved. The challenge $\mathbf{C}_0$ represents the situation where the recognition must fail, because the two parties executing $\rho_{j'}(\overline{\mathbf{const}}_{j'})$ and contributing to the two binding sequences are different. If one can construct a distinguisher that can correctly determine the value of b in C_b then clearly this distinguisher can be used to recognize a party.

Proposition 5.5 *The operational recognition is equivalent to the conceptual recognition.*

Proof. First we consider the forward implication: the operational recognition implies the conceptual recognition. The main idea is to use the distinguisher of

the operational recognition to satisfy the requirement of the conceptual recognition.

For this purpose, on A, we create a local database: $db ::= \{id_n : id_n \in \{0,1\}^{|\beta_j|}\}$. The database db contains binding sequences, and it may grow to N records, where N is the total number of network entities. Initially, the database is empty: $db = \epsilon$. Each time an instance $\beta_j(r)$ is received, the party A does the following.

1. A goes through each entry id_n in db and query the distinguisher $dst^{rcog}($ $(id_n, \beta_j(r)), \overline{\textbf{const}})$.
2. If the distinguisher returns 1 on a value of id_n then this value of id_n is the pseudo identity of the peer entity; this also shows that the peer entity once existed on the network.
3. If the distinguisher returns 0 on all entries then extend the database by including a new pseudo identity: $db \leftarrow db \cup \{\beta_j(r)\}$. In this case, the party A concludes that the peer entity never communicated to A before.

Clearly, if the operational recognition holds then A can decide whether or not A has communicated to the same peer entity before. Therefore, the forward implication holds. Next, we consider the reverse implication: the conceptual recognition implies the operational recognition.

In this case, we can use a contrapositive argument: if a distinguisher for the operational recognition does not exists then there exists an efficient attack against the conceptual recognition. Assuming no distinguisher exists on A, computationally the following two pairs are equal: $(\beta_j(r), \beta_j(r'')) \approx (\beta_j(r), \beta_j(r'))$ and we have $\beta_j(r'') \approx \beta_j(r')$. Therefore, A is oblivious to pseudo identities (binding sequences) of network parties, which is a clear violation of the conceptual definition.
Hence, the reverse implication holds and the operational recognition is equivalent to the conceptual recognition. □

5.2.2 Identification

The operational definition of the identification goal can be constructed in a similar manner as that of the recognition. Once again, the identity of the local party, who identifies a peer entity, is not important for the identification purpose. It is actually a role program that identifies a peer entity, but, for convenience,

we assume that the role program is executed by a party A. The operational definition is as follows.

Definition 5.6 (Identification) Let

1. $\beta_j(r)$ and $\beta_j(r')$ be the two binding sequences that are received in two different runs of a role program $\rho_j(\overline{\mathbf{const}}_j)$.
2. another role program be $\rho_{j'}(\overline{\mathbf{const}}_{j'})$ that sends these binding sequences.
3. the role program $\rho_j(\overline{\mathbf{const}}_j)$ be executed by a party A in the both runs.
4. $\bar{B}$ to be a constant in $\overline{\mathbf{const}}_j$.
5. the role program $\rho_{j'}(\overline{\mathbf{const}}_{j'})$ be executed by B in the first run and by C in the second run.
6. the two authentication challenges be $\mathbf{C}_1 = \beta_j(r)$ and $\mathbf{C}_0 = \beta_j(r')$.

The party A is said to achieve $Idnt(A \triangleright B, \beta_j)$ (the identification of B by A based on β_j) if one of the following conditions holds.

- The challenge $\mathbf{C}_0$ is inconsistent with security assumptions.
- A distinguisher $dst^{idnt}(C_b, \overline{\mathbf{const}}_j)$ can be constructed in $\rho_j(\overline{\mathbf{const}}_j)$ for all choices of B and C.

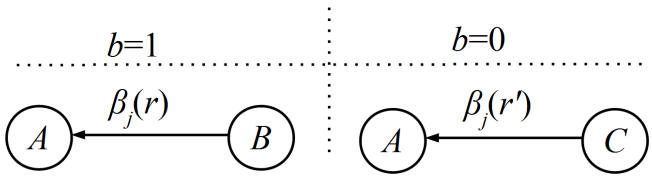


Figure 5.4: Configurations for the Distinguishers of $Idnt(A \triangleright B, .)$

As shown in Fig. 5.4, for $b = 1$ the network configuration is the same as expected by A, while for $b = 0$ the network configuration is not consistent with the expectation of A. Note that $\bar{B}$, in the list of constants, represents the identity of a party whose identification is required. The value of B that is assigned during the execution is a claimed identity of a peer entity. The above definition specifies that in order to achieve the identification of B there must exist a distinguisher that can differentiate the cases when A is communicating with B and when A is communicating with C, for $C \neq B$.

Proposition 5.7 *The operational identification is equivalent to the conceptual identification.*

PROOF.[Sketch] The arguments are essentially the same as used in Proposition 5.5. In the forward implication, the database is already available in form of an identification database. Therefore, if no match is found we reject the claimed identity. For the reverse implication, an additional conclusion is that A does not make an effective use of the identification database. □

5.2.3 Willingness

The willingness goal is concerned with the consent of a peer entity B to communicate with A. In particular, this means that A gets a piece of evidence from which A can conclude that a party who claims to be B wants to communicate with A. Unlike the previous operational definitions, the party A obtains the willingness of another role program (and not of the party who executes the other role program). For convenience, we assume that the other role program is executed by B. The operational definition is as follows.

Definition 5.8 (Willingness) Let

1. the two binding sequences $\beta_j(r)$ and $\beta_j(r')$ are sent in two runs of a role program $\rho_{j'}(\overline{\mathbf{const}}_{j'})$, which is executed by B.
2. the role program $\rho_j(\overline{\mathbf{const}}_j)$ be the receiver of two binding sequences and is executed by A and C in the two runs.
3. $\bar{A}$ be a constant in $\overline{\mathbf{const}}_j$.
4. the two challenges be $\mathbf{C}_1 = \beta_j(r)$ and $\mathbf{C}_0 = \beta_j(r')$.

The party A is said to achieve $Wlng(A \triangleright B, \beta_j)$ (the willingness of B by A based on β_j) if one of the following conditions holds.

- The challenge $\mathbf{C}_0$ is inconsistent with security assumptions.
- A distinguisher $dst^{wlng}(C_b, \overline{\mathbf{const}}_j)$ can be constructed in $\rho_j(\overline{\mathbf{const}}_j)$ for all choices of B and C.

The condition $\bar{A} \in \overline{\mathbf{const}}_j$ means that the role program $\rho_j(\overline{\mathbf{const}}_j)$, which executed by A, knows the identity of A; without knowing the identity of A, $\rho_j(\overline{\mathbf{const}}_j)$ cannot decide whether a peer entity is communicating with A or not.

The two configurations involved in the willingness definition are also shown in Fig. 5.5. For $b = 1$, the role program executed by B is communicating to A, which is the expected behaviour. For $b = 0$, the role program executed by

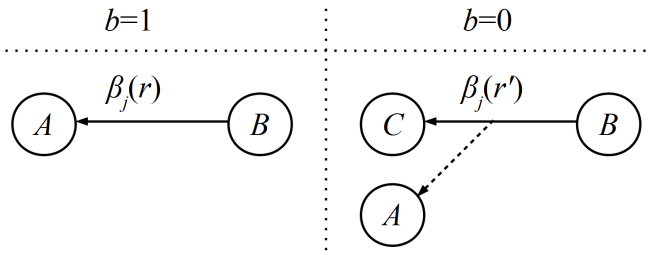


Figure 5.5: Configurations for the Distinguishers of $Wlng(A \triangleright B, .)$

B is communicating to another party C and the resulting binding sequence is being replayed to A. If the role program of A can distinguish between these two configurations then clearly A can conclude about the consent of B.

Proposition 5.9 *The operational willingness is equivalent to the conceptual willingness.*

Proof. First, consider the forward implication: the operational willingness implies the conceptual willingness. When A gets an instance of a binding sequence $\beta_j(r)$ from B then the operational willingness guarantees that $\beta_j(r)$ is specifically sent for A. This information implies that B achieves identification for A before sending his willingness, because, otherwise B cannot send different binding sequences to A and C. Therefore, sending of a binding sequence by B for A implies the willingness of B to communicate with A.

Similarly, the conceptual willingness implies the operational willingness. Once again, we use the corresponding contrapositive argument. If a distinguisher for the willingness does not exist on A then we have $\beta_j(r) \approx \beta_j(r')$, which means that either B cannot identify peer entities or B does not sends its willingness in a manner understandable to A. In any case, the local party A cannot get an assurance that B is willing to communicate with A. Hence, the operational willingness implies the conceptual willingness. □

5.2.4 Operativeness

The operativeness goal is concerned with the current availability of a peer entity. Similar to the definitions of identification and willingness, the operativeness goal is achieved by a role program and not by a party, but for convenience we

assume that a party A executes the role program that achieves operativeness. The operational definition of operativeness is as follows.

Definition 5.10 (Operativeness:) Let

1. β_j be a binding sequence of a role program $\rho_j(\overline{\mathbf{const}}_j)$ that is executed by A.
2. $\mathbf{X} \subseteq \beta_j$ be a list consisting of messages sent by a role program $\rho_{j'}(\overline{\mathbf{const}}_{j'})$ that is claimed to be executed by B.
3. $\bar{N}_A$ be a nonce and is a constant in $\rho_j(\overline{\mathbf{const}}_j)$, i.e., $\bar{N}_A \in \overline{\mathbf{const}}_j$.

If a dependency function can be constructed from $\bar{N}_A$ to X in $\rho_j(\overline{\mathbf{const}}_j)$ then A is said to achieve the operativeness of the party that claims to be B.

Unlike the previous definitions, this definition does not depend on a distinguisher, The operativeness only requires the construction of a D-function. If the required D-function exists then it is not possible to replay the messages in $\mathbf{X}$, because replaying a message implies the existence of a collision, i.e., two different values of the nonce in two different runs map to the same replayed value of $\mathbf{X}$.

A nonce can be instantiated by a random number, time stamp, or sequence number. A true random number is a unpredictable value, which cannot be computed by dishonest parties, while time-stamps or sequence numbers are predictable values. These choices are important in the security analysis that guarantee the canonicity of messages of a binding sequence. Gong [81] provides a detailed account on the use of different types of nonces in authentication protocols.

If a nonce is a random number or a sequence number then the calling routine sends different values of the nonce in different runs. In the case of a time-stamp, operativeness cannot be guaranteed among the runs that are initialized with the same time-stamp.

Proposition 5.11 *The operational operativeness is equivalent to the conceptual operativeness.*

Proof. First consider the forward implication: the operational operativeness implies the conceptual operativeness. The operational operativeness guarantees that for each value of the nonce N_A in Def. 5.10, there is a different binding sequence β_j, because, otherwise, there will be a collision in the dependency function of $N_A \rightarrow \mathbf{X}$, where $\mathbf{X} \subseteq \beta_j$. Therefore, the binding sequence β_j must

be freshly generated by a party who claims to be B, which implies that the party is currently on the network.

Now, we consider the reverse implication. For this purpose, we can use a contrapositive argument: if the D-function required for the operational operativeness in Def. 5.10 does not exist then any old binding sequence can be replayed, which means that a party who claims to be B may not be currently active on the network. This violates the requirement of the conceptual operativeness. Therefore, the conceptual operativeness implies the operational operativeness. □

5.2.5 High-level FLAGs

There are a number of high-level FLAGs in the hierarchy shown in Fig. 5.1. A high-level FLAG depends on one or more low-level FLAGs. Note that a combination of multiple FLAGs may not correspond to a new high-level FLAG, because the FLAGs on which a high-level FLAG depends are all achieved from the same binding sequence.

Definition 5.12 (High-Level FLAGs) The operational definitions of high-level FLAGs are as follows:

(1) One-sided Authentication: $\text{OATH}(A \triangleright B, \beta_j) \stackrel{\text{def}}{=} \mathit{Wlng}(A \triangleright B, \beta_j) \wedge \mathit{Oper}(A \triangleright B, \beta_j) \wedge \mathit{Idnt}(A \triangleright B, \beta_j)$

(2) Pseudo One-sided Authentication: $\text{POATH}(A \triangleright B, \beta_j) \stackrel{\text{def}}{=} \mathit{Wlng}(A \triangleright B, \beta_j) \wedge \mathit{Oper}(A \triangleright B, \beta_j) \wedge \mathit{Recog}(A \triangleright B, \beta_j)$

(3) Confirmation: Let $\beta_j = \mathbf{M}_1 || \mathbf{M}_2$ (|| stands for concatenation). $\mathit{Cnfm}(A \triangleright B, G, \beta_j) \stackrel{\text{def}}{=} \mathit{Idnt}(A \triangleright B, \mathbf{M}_2) \wedge \mathit{Oper}(A \triangleright B, \mathbf{M}_2)$ and G is achieved in the flows corresponding to M_1.

(4) Strong One-sided Authentication: $\text{Str-OATH}(A \triangleright B, \beta_j) \stackrel{\text{def}}{=} G \wedge \mathit{Cnfm}(A \triangleright B, G, \beta_j)$, where $G = \text{OATH}(A \triangleright B, \beta_j)$

(5) Mutual Authentication: $\text{MATH}(A \triangleright B, \beta_j) \stackrel{\text{def}}{=} \text{OATH}(A \triangleright B, \beta_j) \wedge \mathit{Cnfm}(A \triangleright B, G, \beta_j)$, where $G = \text{OATH}(B \triangleright A, .)$

The first high-level FLAG, one-sided authentication, requires that the willingness, operativeness, and identification of B are achieved from the same binding sequence. Note that the party who claims to be B must be the same in the

willingness and operativeness goals, and the same party is then identified as part of achieving the identification goal. Conceptually, one-sided authentication guarantees A that whoever claims to be B is indeed B (due to $Idnt(.)$), B is currently ready to communicate (due to $Oper(.)$), and B indeed wants to communicate with A (due to $Wlng(.)$).

The second high-level FLAG is the same as the first one except it depends on entity recognition instead of entity identification. The third high-level FLAG is the confirmation of another FLAG. Intuitively, A achieves the confirmation of a FLAG G from B if the following conditions hold:

1. If B knows that G has been achieved.
2. After knowing that G has been achieved, B sends a message to A as a part of the same binding sequence in which G was achieved.

These conditions are formalized in the operational definition of confirmation, in which the binding sequence is further divided in two parts. From the first part, A achieves a FLAG G, and, from the from the second part, A obtains the confirmation that B knows that G has been achieved. The next two FLAGS depend on the confirmation goal.

The fourth high-level FLAG is strong one-sided authentication, in which A not only achieves the one-sided authentication of B but also confirms that B knows that A has achieved this goal. The fifth high-level FLAG is mutual authentication. Mutual authentication occurs when two parties A and B authenticate each other. From A's view point, the knowledge of the authentication of A by B is obtained by the confirmation goal.

All of the operational definitions are constructive in nature, namely each definition specifies a requirement for the construction of a concrete distinguisher function (or a dependency function). If a security analyst is not able to construct a required distinguisher function then this does not necessarily mean that the corresponding FLAG cannot be achieved. It only means that the security analysis is not able to find a way to satisfy the definition. It may still be possible that someone else finds a way to construct the required distinguisher to show that the FLAG is achieved.

One should understand the meaning of achieving a FLAG in the following sense:

- A security analyst carries out the correctness analysis, which determines that a certain FLAG is achieved by a role program.
- Now, if a party A executes the role program, which results in a successful execution of the program, then A is said to achieve that FLAG.

This view point is captured in the following definition of a successful run.

Definition 5.13 (Successful Run) A DC-run is called a successful run if all of the distinguishers implemented in the role program return 1.

It may be the case that a role program does not have any distinguisher implementations (see operational definitions of FLAGs). Therefore, dependency compliance (DC) may be sufficient for a successful run.

The term entity authentication can refer to any subset of FLAGs. The meaning of correctness of an authentication protocol, therefore, depends on the sets of FLAGs that the role programs of the protocol achieve. Even if we consider a simple two-role protocol along with a set of three FLAGs (e.g., identification, willingness, and operativeness), there are sixty three (63) possible interpretations of entity authentication[1].

As different protocols may achieve different sets of FLAGs, different applications, which use entity authentication as a service, may also require different sets of FLAGs. For example, bar-code (or RFID tag) based authentication of goods at a payment counter may only require the identification of a tag, while the authentication used for on-line banking usually requires the identification (using a user-name and a password), operativeness (using a user-specific challenge) and willingness (by accepting a certificate for the bank's website) of an account holder. Therefore, it is important that a system developer correctly identifies the required set of FLAGs for his application[2], and a security analyst correctly determines the set of FLAGs that a protocol achieves. A mismatch between achieved FLAGs and required FLAGs may lead to security vulnerabilities even when the protocol is proved to be secure.

5.3 Case Study 1

We continue with our example of ISO-9798 five-pass authentication protocol from the previous chapter. The protocol consists of three role programs. For

[1]There are eight combinations of three FLAGs. Therefore, a role program can achieve any eight of these combinations. A two-role protocol consists of two role programs. Each combination of FLAGs of a role program corresponds to eight different combinations of FLAGs for the other role program. This means that there are 8×8 combinations for the protocol. One of these 64 combinations corresponds to the case when both role programs achieve no FLAG. Excluding this combination results in 63 possible interpretations of entity authentication for a two-role protocol.

[2]This is a type of pleasantness problem [65].

each role program, we use the operational definitions of FLAGs to carry out the correctness analysis of the program. The steps involved in our analysis are as follows:

- For identification or willingness:
  1. We identify the two distinguisher challenges: C_1 for an expected configuration, and C_0 for an alternate configuration.
  2. We construct a distinguisher that finds the value of b on the input C_b.
  3. We provide arguments supporting that the construction of the distinguisher is correct, i.e., it indeed generates the correct value of b.
- For the operativeness of B by A (or A by B):
  1. We identify a nonce in the role program executed by A.
  2. We show that there is a dependency arc from the nonce to a message sent by B, and the sent message is in the binding sequence of A.

We do not analyse the server program of the protocol, because the server program does not have any binding sequence. The analyses of the initiator and responder role programs are as follows.

Initiator program: $A^\rho \stackrel{\text{def}}{=} \rho_1(\bar{R}_A, \bar{K}_{AS}, \bar{A}, \bar{B})$

Recall that a *bar* on a term indicates that it is a constant that is supplied to the role program by the calling routine at the start of an execution. Following facts about A^ρ are known:

1. From the previous chapter, we know that the binding sequence of A^ρ is $\beta_1 = [M_6, M_7]$, where $M_6 = \mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B)$ and $M_7 = \mathcal{E}_{K_{AB}}(R_B \| R_A)$.
2. The program A^ρ knows that it is communicating with a responder program B^ρ, due to the definition of a binding sequence. A binding sequence is always sent by expected role programs, which in this case are S^ρ for M_6 and B^ρ for M_7.
3. Without further interpretation of β_1, the program A^ρ does not know whether B^ρ is being executed by B or by another party C, for $C \neq B$. In particular, C may be an adversary who is pretending to be B.

For the identification goal, $\mathit{Idnt}(A \triangleright B, \beta_1)$, the program A^ρ must be able to identify the party who claims to be B in β_1. The identity of B is supplied to A^ρ as a part of the program input.
The two identification challenges are as follows:

$$\mathbf{C}_1 = [\mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B), \mathcal{E}_{K_{AB}}(R_B \| R_A)]$$
$$\mathbf{C}_0 = [\mathcal{E}_{K_{AS}}(R_A \| K_{AC} \| C), \mathcal{E}_{K_{AC}}(R_C \| R_A)] \text{ where } C \neq B$$

The first challenge C_1 represents the expected configuration when A is indeed communicating with B. The alternate challenge C_0 occurs in a configuration when A is not communicating with B. In C_0, we replace K_{AB} with K_{AC}, where $K_{AB} \neq K_{AC}$, to specify that these are two different values. Similarly, we replace B with C to specify that these two are different values. An adversary selects the values of b and presents the challenge C_b to A^ρ. The purpose of the identification distinguisher, which is constructed within A^ρ, is to compute the right value of b, thereby identifying B.

We address the elements of C_b using $C_b[1]$ and $C_b[2]$, e.g., if $b = 1$ then $C_b[1] = \mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B)$ and if $b = 0$ then $C_b[1] = \mathcal{E}_{K_{AS}}(R_A \| K_{AC} \| C)$. The main idea used in the identification distinguisher is that A^ρ receives an authentic message from B as a part of β_1, which implies that B is identified by A^ρ. A construction of the identification distinguisher is as follows:

$dst^{idnt}(C_b, \bar{K}_{AS}, \bar{B}, \bar{R}_A)$:
(1) $x_1, x_2, x_3 \leftarrow \mathcal{D}_{\bar{K}_{AS}}(C_b[1])$
(2) $x_4, x_5 \leftarrow \mathcal{D}_{x_2}(C_b[2])$
(3) If $x_3 = \bar{B} \wedge x_5 = \bar{R}_A$ return 1
(4) Else return 0

In the distinguisher, if the condition $x_3 = \bar{B}$ holds then A concludes that the decrypted key x_2 is for the communication between A and B. This conclusion is implied by the setup assumptions of the protocol, namely there is a secure channel between A and S defined by K_{AS}, and the reply from S includes the information that who else knows the newly generated session key. If the second condition $x_5 = \bar{R}_A$ also holds then it means that it is B that has sent this message, because only B is the peer entity that knows the session key x_2. Therefore, if the above distinguisher returns 1 then A concludes that B once existed on the network.

Note that the functions that are part of this distinguisher are also part of the dependency graph of A^ρ. This means that there is not need to implement this distinguisher separately, because the verification of the dependency functions succeed only if $b = 1$, where $b = 1$ means A is communicating with B as expected.

Next, we consider the operativeness goal, which requires the existence of a dependency arc from the nonce of A^ρ to a message of β_1 that is sent by B^ρ. This goal is achieved, because there is a dependency arc from R_A to $\mathcal{E}_{K_{AB}}(R_B \| R_A)$, which was shown in Fig. 4.11 on page 83. The message variable $\mathcal{E}_{K_{AB}}(R_B \| R_A)$ is in the binding sequence β_1. Therefore, the goal $\mathit{Oper}(A \triangleright B, \beta_1)$ is achieved.

Next, we consider the willingness goal, $\mathit{Wlng}(A \triangleright B, \beta_1)$. The message $\mathcal{E}_{K_{AB}}(R_B \| R_A)$ in β_1 is sent by B. The two willingness challenges are as follows:

$$\begin{aligned}\mathbf{C}_1 =& [\mathcal{E}_{K_{AS}}(R_A \| K_{AB} \| B), \mathcal{E}_{K_{AB}}(R_B \| R_A)] \\ \mathbf{C}_0 =& [\mathcal{E}_{K_{AS}}(R_C \| K_{CB} \| B), \mathcal{E}_{K_{CB}}(R_B \| R_C)] \quad \text{where } C \neq A\end{aligned}$$

The first challenge C_1 represents the expected configuration when B is indeed communicating with A. The alternate challenge C_0 occurs in a configuration when B is communicating with a party C and an adversary forwards the communicated messages to A. The challenge C_b is presented to A^ρ, in which the value of b is selected by an adversary. The purpose of the willingness distinguisher, which is constructed within A^ρ, is to compute the right value of b. The distinguisher for the willingness is the same as that of the identification goal. This is because, if the second condition $x_5 = \bar{R}_A$ holds then it means that it is B that has sent M_7 specifically for A. Therefore, the goal $\mathit{Wlng}(A \triangleright B, \beta_1)$ is also achieved.

The three low level FLAGs are achieved from the same binding sequence. This satisfies the requirement of one-sided authentication:

$$\text{OATH}(A \triangleright B, \beta_1) = \mathit{Wlng}(A \triangleright B, \beta_1) \wedge \mathit{Oper}(A \triangleright B, \beta_1) \wedge \mathit{Idnt}(A \triangleright B, \beta_1)$$

Therefore, the initiator role program achieves one-sided authentication of the party that is executing the responder role program.

Responder program: $B^\rho \stackrel{\text{def}}{=} \rho_2(\bar{R}_B, \bar{R}'_B, \bar{K}_{BS}, \bar{A}, \bar{B})$

The analysis for B^ρ is similar to that of A^ρ. From the previous chapter, we know that the binding sequence of B^ρ is $\beta_2 = [M_1, M_4, M_8]$, where $M_1 = R_A$, $M_4 = \mathcal{E}_{K_{BS}}(R'_B \| K_{AB} \| A)$, and $M_8 = \mathcal{E}_{K_{AB}}(R_A \| R_B)$.

First we consider the identification goal $\mathit{Idnt}(B \triangleright A, \beta_2)$. The two identification challenges are as follows:

$$\mathbf{C}_1 = [R_A, \mathcal{E}_{K_{BS}}(R'_B \| K_{AB} \| A), \mathcal{E}_{K_{AB}}(R_A \| R_B)]$$
$$\mathbf{C}_0 = [R_C, \mathcal{E}_{K_{BS}}(R'_B \| K_{CB} \| C), \mathcal{E}_{K_{CB}}(R_C \| R_B)]$$

The first challenge C_1 represents the expected configuration when B is indeed communicating with A. The alternate challenge C_0 occurs in a configuration when B is not communicating with A. An adversary selects the values of b and the challenge C_b is presented to the role program B^ρ. The distinguisher for the identification is as follows.

$dst^{idnt}(C_b, \bar{K}_{BS}, \bar{A}, \bar{R'}_B, \bar{R}_B)$
(1) $x_1, x_2, x_3 \leftarrow \mathcal{D}_{\bar{K}_{BS}}(C_b[2])$
(2) $x_4, x_5 \leftarrow \mathcal{D}_{x_2}(C_b[3])$
(2) If $x_3 = \bar{A} \wedge x_5 = \bar{R}_B$ return 1
(3) Else return 0

This distinguisher uses the condition $x_3 = \bar{A}$ to determine that x_2 is the key that is only known to A and B. The verification $x_5 = \bar{R}_B$ assures that A has sent a message, which is sufficient to conclude that A exists on the network.

Next, we consider the goal of the operativeness of A by B^ρ, for which it is required that a dependency function exists from a nonce of B^ρ to a variable in the binding sequence of B^ρ. The term $\bar{R}_B$ is a nonce that is mapped to $\mathcal{E}_{K_{AB}}(R_A \| R_B)$ by a dependency function, and the term $\mathcal{E}_{K_{AB}}(R_A \| R_B)$ is sent by A. Therefore, the goal $Oper(B \rhd A, .)$ is achieved.
Next, we consider the willingness goal, for which the two challenges are as follows.

$$\mathbf{C}_1 = [R_A, \mathcal{E}_{K_{BS}}(R'_B \| K_{AB} \| A), \mathcal{E}_{K_{AB}}(R_A \| R_B)]$$
$$\mathbf{C}_0 = [R_A, \mathcal{E}_{K_{BS}}(R'_C \| K_{AC} \| A), \mathcal{E}_{K_{AC}}(R_A \| R_C)]$$

The first challenge C_1 represents the expected configuration when A is communicating with B. The alternate challenge C_0 occurs in a configuration when A is not communicating with B, instead A is communicating with C, for $C \neq B$. The distinguisher for the willingness goal is the same as that for the identification. The distinguisher is correct for the willingness goal, because only A and B knows the value of x_2, and therefore $M_8 = \mathcal{E}_{K_{AB}}(R_A \| R_B)$ is specifically sent for B by A.

Since the three low level FLAGs are achieved from the same binding sequence, we conclude that B achieves one-sided authentication of A:

$$\text{OATH}(B \rhd A, \beta_2) = \mathit{Wlng}(B \rhd A, \beta_2) \wedge \mathit{Oper}(B \rhd A, \beta_2) \wedge \mathit{Idnt}(B \rhd A, \beta_2)$$

This completes the correctness analysis.

5.4 Summary

In this chapter, we described the third step of the SI methodology, which is the correctness analysis. We introduced the notion of fine level authentication goals (FLAGs) and provided two types of definitions for each FLAG. First, we presented conceptual definitions, which define FLAGs from a service-oriented perspective, in a protocol independent manner. Recognition, identification, operativeness, and willingness are low level FLAGs, because they do not depend on other FLAGs. There is a partial order between different FLAGs, and therefore FLAGs constitute a hierarchy of authentication properties. We presented the operational definitions of FLAGs, which provide operational interpretations of FLAGs in terms of a binding sequence of a role program. The conceptual definition and the operational definition of a FLAG are shown to be equivalent. Operational definitions of FLAGs are constructive in nature, namely in order to show that a role program achieves a certain FLAG, a security analyst constructs a distinguisher function in the role program. The derivation process of a FLAG is a non-security process, since the binding sequence is used as an assumption, which means that one does not need to consider the dynamic behaviour of the role program in the presence of a network adversary. At last, we carried out the correctness analysis of the protocol of our case study.

Chapter 6

Insecure Protocols

In this chapter, we analyse two protocols that are insecure in a traditional sense. The purpose is to show that the qualification of being secure or insecure actually depends on what one expects from a protocol.

The first protocol is the Needham-Schroeder public-key (NSPK) protocol [117], which was proposed in 1978. It was believed to be secure until 1996 when Lowe discovered an attack against the protocol. The second one is the Woo-Lam entity authentication (WL) protocol [156], for which Abadi [158] discovered an attack.

6.1 NSPK Protocol

The Needham-Schroeder public-key (NSPK) protocol [117] is a three-role protocol and consists of seven flows. Out of the seven flows, four flows are used to obtain the public keys of protocol parties from a trusted server. The protocol parties can skip these four flows if, e.g., they have already cached the public keys of each other.

The remaining three flows constitute a two-role protocol, in which it is assumed

that protocol parties are already in possession of each other's correct public keys. The protocol narration is as follows.

$$\begin{array}{lll} (1) & A^{\rho} \rightarrow B^{\rho}: & M_1 = \mathcal{E}_{Pk_B}(R_A \| A) \\ (2) & B^{\rho} \rightarrow A^{\rho}: & M_2 = \mathcal{E}_{Pk_A}(R_A \| R_B) \\ (3) & A^{\rho} \rightarrow B^{\rho}: & M_3 = \mathcal{E}_{Pk_B}(R_B) \end{array}$$

The protocol consists of two role programs executed by two parties A and B. In the protocol, R_A and R_B are two random numbers generated by A and B respectively. The party A executes the initiator program:

$$A^{\rho} \stackrel{\text{def}}{=} \rho_1(\bar{P}k_B, \bar{S}k_A, \bar{R}_A, \bar{B})$$

The party B executes the responder program:

$$B^{\rho} \stackrel{\text{def}}{=} \rho_2(\bar{P}k_A, \bar{S}k_B, \bar{R}_B, \bar{A})$$

Because the NSPK protocol is a two-role protocol, the role narrations of the two programs are the same as the protocol narration. Also, the global D-graph of the protocol and the local D-graphs of the role programs are the same, which are denoted by $\mathbf{D}_{nspk}$, as shown in Fig. 6.1. The D-graph consists of three dependency nodes, M_1, M_2, and M_3, and two interim nodes, R_A and R_B. The dependency functions of $\mathbf{D}_{nspk}$ are clear from the protocol narration.

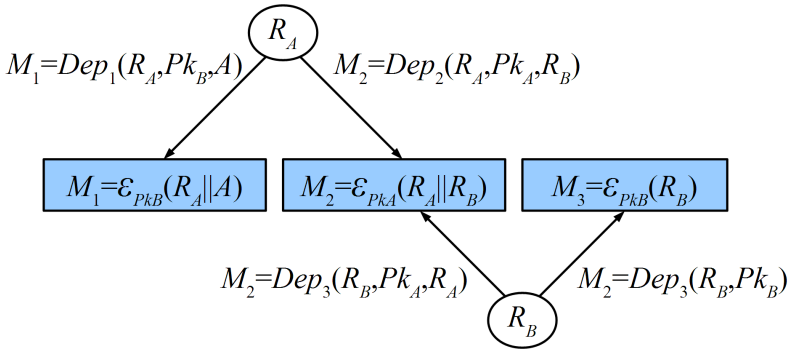


Figure 6.1: Global and Local D-graphs of NSPK protocol: $\mathbf{D}_{nspk}$

6.1.1 Initiator role program $\rho_1(\bar{P}k_B, \bar{S}k_A, \bar{R}_A, \bar{B})$

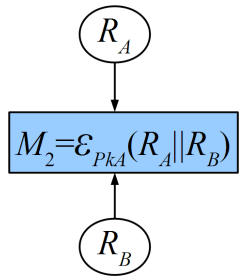


Figure 6.2: The graph $\vec{in}(2, \mathbf{D}_{nspk})$

Since $\rho_1(.)$ only communicates with the role program $\rho_2(.)$, the two subgraphs $\vec{in}(2, \mathbf{D}_{nspk})$ and $\vec{in}(*, \mathbf{D}_{nspk})$, which are used to compute the binding sequence of $\rho_1(.)$, are the same. The graph $\vec{in}(2, \mathbf{D}_{nspk})$ is shown in Fig. 6.2, in which there is only one dependency node $M_2 = \mathcal{E}_{Pk_A}(R_A \| R_B)$. For the security analysis, it is required that the message variable M_2 is always assigned with a canonical message. A canonical message for M_2 is a value that is sent by the role program $\rho_2(.)$ in the second flow. We present an informal analysis in the following.

We claim that the canonicity requirement is met by the protocol, because only B is the peer entity who knows the value of R_A. There is no other message in the protocol that is type-compatible with M_2, therefore typing errors cannot occur. Therefore, it must be either A or B executing B^ρ who has sent the message for M_2. This also means that we must require R_A to be an unpredictable random number so that a network adversary cannot compute it beforehand. The use of other types of nonces, such as a sequence number, may not suffice to satisfy the canonicity requirement. The resulting binding sequence is as follows (as per Proposition 4.7):

$$\beta_1 = node(\vec{in}(*, \mathbf{D}_{nspk})) = M_2 = \mathcal{E}_{Pk_A}(R_A \| R_B)$$

Now, we start the correctness analysis to determine which FLAGs are achieved by A that executes the initiator role program. Although the NSPK protocol is relatively small as compared to our earlier case study, it is more intricate in terms of correctness analysis due to the use of public key encryption. The reader may want to revise our earlier discussion on public key encryption on Pg. 67.

For the identification FLAG, $Idnt(A \triangleright B, \beta_1)$, the program A^ρ identifies a party who executes B^ρ and claims to be B. The identity of B is supplied to A^ρ as

a part of its input. The program A^ρ knows that it is communicating with a responder program B^ρ, due to the binding sequence of A^ρ, which consists of a message from B^ρ. The program A^ρ, however, does not know whether B^ρ is being executed by B or by another party C, for $C \neq B$.

As per the operational definition of identification (Def. 5.6 on Pg. 100), the two identification challenges are as follows (see Fig. 5.4 on Pg. 100):

$$\begin{aligned}\mathbf{C}_1 &= \mathcal{E}_{Pk_A}(R_A \| R_B) \quad \text{(Expected configuration: } B \longrightarrow A\text{)} \\ \mathbf{C}_0 &= \mathcal{E}_{Pk_A}(R_A \| R_C) \quad \text{(Alternate configuration: } C \longrightarrow A\text{)}\end{aligned}$$

The first challenge is generated in a network configuration that is expected by A, namely B^ρ is indeed executed by A. The second challenge is generated when there is an unexpected configuration, namely B^ρ is not executed by A. Clearly, it is not possible to construct a distinguisher, because the value of R_A is the same in both of the challenges.

Therefore, the only way A^ρ can achieve the identification goal, as per the operational definition, is to show that the alternate configuration cannot occur, namely the alternate configuration is inconsistent with security assumptions. For this purpose, we note that $\mathbf{C}_0$ implies that the party C knows the value of R_A. Since R_A is a randomly generated value, it is unpredictable. Assuming that the private key of B is secret, the value of R_A is only disclosed to B. Therefore, the alternate configuration cannot occur and we conclude $\mathit{Idnt}(A \rhd B, \beta_1)$ is achieved.

For the operativeness goal, we use the operational definition Def. 5.10. We note that there is a dependency arc from R_A to $\mathcal{E}_{Pk_A}(R_A \| R_B)$, which meets the requirement of Def. 5.10. Therefore, we conclude that A achieves the operativeness goal for B: $\mathit{Oper}(A \rhd B, \beta_1)$.

Next, we consider the willingness goal, $\mathit{Wlng}(A \rhd B, \beta_1)$, To validate $\mathit{Wlng}(A \rhd B, \beta_1)$, we use the operational definition Def. 5.8. The first willingness challenge $\mathbf{C}_1$ occurs when B believes that it is communicating with A. The second challenge occurs when B believes that it is communicating with C (see Fig. 5.5). The resultant challenges are as follows:

$$\begin{aligned}\mathbf{C}_1 &= \mathcal{E}_{Pk_A}(R_A \| R_B) \quad \text{(Expected configuration: } B \longrightarrow A\text{)} \\ \mathbf{C}_0 &= \mathcal{E}_{Pk_A}(R_A \| R_B) \quad \text{(Alternate configuration: } B \longrightarrow C\text{)}\end{aligned}$$

Note that, in $\mathbf{C}_0$, we use R_A instead of R_C, because in both challenges A is

the communicating partner, and B only believes that C is the communication partner. Clearly, we cannot construct a distinguisher for the willingness goal, because the two challenges are equal. Now, we explore the possibility whether $\mathbf{C}_0$ is inconsistent with our security assumptions. The challenge $\mathbf{C}_0$ implies that B believes that it is communicating to C, which means that B has received a message in the first step of the form $\mathcal{E}_{Pk_B}(R_A \| C)$. This message can only be generated if C or an adversary knows R_A. Since R_A is an unpredictable nonce, therefore we conclude that $\mathbf{C}_0$ cannot occur. Hence, the willingness of B is achieved by A.

Since $Idnt(A \rhd B, \beta_1)$, $Oper(A \rhd B, \beta_1)$, and $Wlng(A \rhd B, \beta_1)$ are achieved from the same binding sequence, we conclude that A achieves one-sided authentication of B: $\mathrm{OATH}(A \rhd B, \beta_1)$.

6.1.2 Responder role program $\rho_2(\bar{Pk}_A, \bar{Sk}_B, \bar{R}_B, \bar{A})$

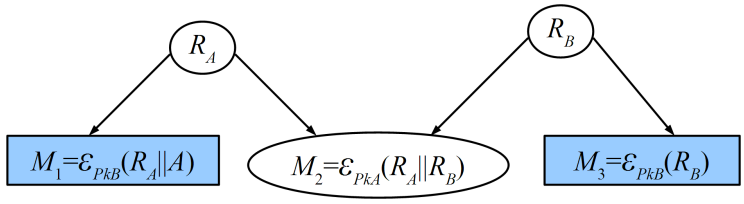


Figure 6.3: The graph $\vec{in}(1, \mathbf{D}_{nspk})$

The role program B^ρ only communicates with A^ρ, therefore, the two subgraphs $\vec{in}(1, \mathbf{D}_{nspk})$ and $\vec{in}(*, \mathbf{D}_{nspk})$ are the same, as shown in Fig. 6.3. There are two dependency nodes, M_1 and M_3, which are required to be assigned with canonical messages. Note that the node M_2 is an interim node in the graph $\vec{in}(*, \mathbf{D}_{nspk})$, because it is only used to connect nodes of the graph, but it is not a part of the graph as per the definition of the function $\vec{in}(.,.)$.

The variable M_3 is always assigned by a canonical message, because it is only A and B who know the value of R_B. Therefore, it must be either A or B executing $\rho_1(.)$ who has sent the message for M_3. The nonce R_A must be an unpredictable random number for the canonicity of M_3, otherwise an adversary can compute a valid value for M_3.

The other dependency node in $\vec{in}(1, \mathbf{D}_{nspk})$ is M_1, which does not readily meet the canonicity requirement. This is because, an adversary may also be able to

send a message for M_1 while pretending to be A; the value of R_A in M_1 is not known to B who executes $\rho_2(.)$. The resulting binding sequence is as follows:

$$\beta_2 = M_3 = \mathcal{E}_{Pk_B}(R_B)$$

Now, we carry out the correctness analysis to determine which FLAGs are achieved by B. For the identification goal, $\mathit{Idnt}(B \triangleright A, \beta_2)$, the two identification challenges are as follows

$$\begin{aligned} \mathbf{C}_1 =& \mathcal{E}_{Pk_B}(R_B) \quad \text{(Expected configuration: } A \longrightarrow B) \\ \mathbf{C}_0 =& \mathcal{E}_{Pk_B}(R_B) \quad \text{(Alternate configuration: } C \longrightarrow B) \end{aligned}$$

Once again, it is not possible to construct a distinguisher, but we note that the challenge $\mathbf{C}_0$ is inconsistent with our assumptions, because $\mathbf{C}_0$ implies that the party C knows the value of R_B. Since R_B is a randomly generated value, it is unpredictable. Assuming that the private key of A is secret, the value of R_B is only disclosed to A. Therefore, we conclude that $\mathit{Idnt}(B \triangleright A, \beta_2)$ is achieved.

The operativeness FLAG is also achieved, because there is a dependency arc from R_B to $\mathcal{E}_{Pk_B}(R_B)$. Therefore, we conclude that the goal $\mathit{Oper}(B \triangleright A, \beta_2)$ is achieved.

Next, we consider the willingness goal, $\mathit{Wlng}(B \triangleright A, \beta_2)$. The first willingness challenge $\mathbf{C}_1$ occurs when A believes that it is communicating with B. The second challenge occurs when A believes that it is communicating with C. Again, the resultant challenges are exactly the same:

$$\begin{aligned} \mathbf{C}_1 =& \mathcal{E}_{Pk_B}(R_B) \quad \text{(Expected configuration: } A \longrightarrow B) \\ \mathbf{C}_0 =& \mathcal{E}_{Pk_B}(R_B) \quad \text{(Alternate configuration: } A \longrightarrow C) \end{aligned}$$

Therefore, we cannot construct a distinguisher for the willingness goal. Now, we explore the possibility whether $\mathbf{C}_0$ is inconsistent with our security assumptions. The challenge $\mathbf{C}_0$ implies that A believes that it is communicating to C. To avoid this configuration, there must be some message sent to A from which A learns that that the nonce R_B is from B. This information is never conveyed to A in the protocol. Therefore, we cannot rule out the possibility of computation of $\mathbf{C}_0$ by an adversary.

This completes our analysis of the protocol. Note that in the analysis of willingness we fail to show that the willingness goal is achieved. The SI methodology

does not produce a counter-example, namely a concrete attack trace that shows that an adversary computes $\mathbf{C}_0$. Interestingly, the Lowe's attack [96] on the protocol is a counter example that computes $\mathbf{C}_0$; this attack does not violate the validity of any other FLAGs that are achieved by the two role programs. In the following, we describe the Lowe's attack in detail.

6.1.3 Lowe's attack

We discuss the insecurity caused by the Lowe's attack from the perspective of the SI methodology. In the attack, it is assumed that an adversary is a legitimate user of the network. The narration of the Lowe's attack [96] is listed below, in which $\mathcal{I}^\rho$ represents the attack program that is executed by a dishonest party $\mathcal{I}$.

$$
\begin{array}{lll}
(1) & A^\rho \longrightarrow \mathcal{I}^\rho: & \mathcal{E}_{Pk_{\mathcal{I}}}(R_A \| A) \\
(1') & \mathcal{I}^\rho \longrightarrow B^\rho: & \mathcal{E}_{Pk_B}(R_A \| A) \\
(2) & B^\rho \longrightarrow \mathcal{I}^\rho: & \mathcal{E}_{Pk_A}(R_A \| R_B) \\
(2') & \mathcal{I}^\rho \longrightarrow A^\rho: & \mathcal{E}_{Pk_A}(R_A \| R_B) \\
(3) & A^\rho \longrightarrow \mathcal{I}^\rho: & \mathcal{E}_{Pk_B}(R_B) \\
(4') & \mathcal{I}^\rho \longrightarrow B^\rho: & \mathcal{E}_{Pk_B}(R_B)
\end{array}
$$

In flow (1), the role program A^ρ sends a message to the responder program B^ρ supposedly executed by $\mathcal{I}$. This message is as per the NSPK protocol. The party $\mathcal{I}$ is dishonest, so he does not follow the protocol, namely $\mathcal{I}$ executes his attack program $\mathcal{I}^\rho$ instead of B^ρ. Since $\mathcal{I}$ is dishonest, the FLAGs achieved by A for $\mathcal{I}$ are not relevant to the protocol security.

In flow $(1')$, $\mathcal{I}$ pretends to be A and sends a message to B; the message is the same as in flow (1) but now it is encrypted using B's public key. In flow (2), B replies to $\mathcal{I}$ assuming it to be A. In flow $(2')$, B's reply in third flow is forwarded to A by the attack program. In the last two flows, the reply from A is forwarded to B. In this way, B completes a run of B^ρ assuming that it was communicating with an honest party A.

This attack is a valid attack only if

- it is claimed that B achieves a FLAG G for A.
- the FLAG G is not achieved by B in actuality.

Our analysis shows that the responder program B^ρ executed by B achieves the identification and operativeness goals for A who is executing A^ρ. The Lowe's

attack does not refute the validity of these goals. The attack does not work if A does not participate in an attack session, therefore the identification of A is indeed achieved. The adversary $\mathcal{I}$ cannot replay A's messages to repeat this attack. Every time the attack program is executed, A must be present in the attack session. Therefore, the operativeness goal is achieved.

The Lowe's attack in fact shows that B does not achieve the willingness of A, because, in the attack, A believes that he is communicating with $\mathcal{I}$. Also note that the attack does not violate the definition of our binding sequence. The binding sequence of B^ρ, $\mathcal{E}_{Pk_B}(R_B)$, was indeed sent by the correct role program A^ρ.

The above discussion shows the difference between the results that are obtained in a traditional security analysis and in the SI based analysis. Traditionally, the attack on the NSPK protocol implies that the NSPK protocol is insecure. On the other hand, our analysis provides a fine-grained picture: the NSPK protocol is secure if one expects the following FLAGs:

1. One sided authentication of B, $\mathrm{OATH}(A \rhd B, \beta_1)$
2. Identification of A, $\mathrm{Idnt}(B \rhd A, \beta_2)$
3. Operativeness of A, $\mathrm{Oper}(B \rhd A, \beta_2)$

The protocol is insecure if one expects additional FLAGs, such as willingness of A and strong one-sided authentication of B. Lowe also improves the NSPK protocol, resulting in the following narration.

$$
\begin{array}{lll}
(1) & A^\rho \rightarrow B^\rho: & M_1 = \mathcal{E}_{Pk_B}(R_A \| A) \\
(2) & B^\rho \rightarrow A^\rho: & M_2 = \mathcal{E}_{Pk_A}(R_A \| R_B \| B) \\
(3) & A^\rho \rightarrow B^\rho: & M_3 = \mathcal{E}_{Pk_B}(R_B)
\end{array}
$$

With the new message $M_2 = \mathcal{E}_{Pk_A}(R_A \| R_B \| B)$, the willingness of A is achieved by B^ρ, because now the willingness challenge $\mathbf{C}_0$ is inconsistent with protocol assumptions. The party A knows who has sent R_B and therefore its reply to B conveys the willingness of A to communicate with B.

6.2 Woo-Lam Authentication Protocol

Woo and Lam [156] devised a protocol to achieves unilateral authentication, which in the SI methodology translates to the identification and operativeness[1] goals for a peer entity. The protocol narration is as follows.

$$
\begin{array}{lll}
(1) & A^\rho \to B^\rho\colon & M_1 = A \\
(2) & B^\rho \to A^\rho\colon & M_2 = N_B \\
(3) & A^\rho \to B^\rho\colon & M_3 = \mathcal{E}_{K_{AS}}(N_B) \\
(4) & B^\rho \to S^\rho\colon & M_4 = \mathcal{E}_{K_{BS}}\big(A, \mathcal{E}_{K_{AS}}(N_B)\big) \\
(5) & S^\rho \to B^\rho\colon & M_5 = \mathcal{E}_{K_{BS}}(N_B)
\end{array}
$$

The protocol assumes that each network party shares a long term key with an honest server S. The protocol is a three-role protocol, consisting of three role programs A^ρ, B^ρ, and S^ρ, which are executed by A, B, and S respectively. The correctness requirement is that the party A authenticates itself to the party B.

In flow (1), A sends his identity to B claiming that "I am A." In flow (2), B sends a nonce to A, which A encrypts using his long term key and sends the ciphertext back to B in flow (3). In flow (4), B asks the server S to decrypt the ciphertext along with the information that which key must be used in the decryption. In the last flow, S returns the nonce back to B, which is encrypted by B's long term key. If the decrypted nonce is the same as sent by B then B is said to authenticate A.
In an actual implementation, one can expect that this narration is extended so that A receives an acknowledgement indicating whether the authentication was successful.

The program A^ρ is not designed to achieve any authentication goal. Therefore, we only need to consider the role program B^ρ:

$$B^\rho \stackrel{\text{def}}{=} \rho_2(\bar{K}_{BS}, \bar{N}_B)$$

The program B^ρ participates in each flow, therefore, the role narration is the same as the protocol narration. The local D-graph of B^ρ is denoted by $\mathbf{D}_{wl}$ and is shown in Fig. 6.4. The dependency functions of the D-graphs are clear from the protocol narration. Note that there is no arc between M_2 and M_3, because B^ρ does not know K_{AS} and cannot verify $M_2 \to M_3$.

[1] Although the term operativeness was not used by Woo and Lam, they devised the protocol to withstand replay attacks by employing a nonce term, which is essentially the same concept as operativeness.

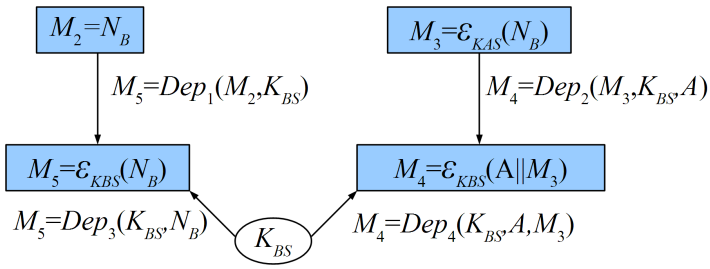


Figure 6.4: The local D-graph of B^ρ: $\mathbf{D}_{wl}$

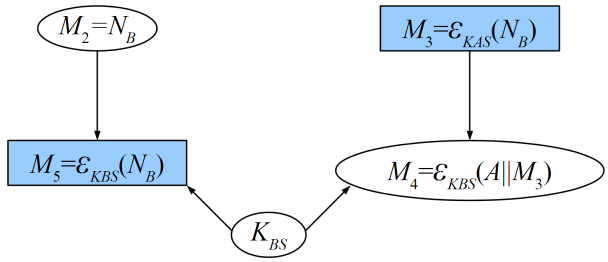


Figure 6.5: The graph $\vec{in}(*, \mathbf{D}_{wl})$

The two subgraphs $\vec{in}(1, \mathbf{D}_{wl})$ and $\vec{in}(*, \mathbf{D}_{wl})$ are the same, as shown in Fig. 6.5. The graph $\vec{in}(*, \mathbf{D}_{wl})$ consists of two dependency nodes $M_3 = \mathcal{E}_{K_{BS}}(N_B)$ and $M_5 = \mathcal{E}_{K_{BS}}(N_B)$. A received message for M_3 cannot be canonical, because B^ρ does not know K_{AS} and therefore cannot verify the message. The value for M_5 is always a canonical message, because this is the only message that the role program S^ρ sends in the protocol and is encrypted by B's long term key, which prevents an adversary from creating the message. We conclude that the binding sequence of B^ρ is $\beta_2 = \mathcal{E}_{K_{BS}}(N_B)$.

6.2.1 Correctness Analysis

Now, we start the correctness analysis to find out which FLAGs are achieved by the role program B^ρ using β_2. We start with the identification FLAG. The two identification challenges are as follows.

$$\begin{aligned}\mathbf{C}_1 =& \mathcal{E}_{K_{BS}}(N_B) \quad \text{(Expected configuration: } A \longrightarrow B\text{)}\\ \mathbf{C}_0 =& \mathcal{E}_{K_{BS}}(N_B) \quad \text{(Alternate configuration: } C \longrightarrow B\text{)}\end{aligned}$$

Since we have $\mathbf{C}_1 = \mathbf{C}_0$, it is not possible to construct a distinguisher for the identification. We now analyse whether $\mathbf{C}_0$ is inconsistent with any of the protocol assumptions. The challenge $\mathbf{C}_0$ implies that that B^ρ has previously sent a message of the form $\mathcal{E}_{K_{BS}}\big(C \| \mathcal{E}_{K_{CS}}(N_B)\big)$. Sending such a message implies that B previously received $\mathcal{E}_{K_{CS}}(N_B)$, which is possible if C is a dishonest party. The challenge $\mathbf{C}_0$ can be computed in the alternate configuration, and therefore the identification goal cannot be achieved.

For the operativeness FLAG, there is a dependency arc from N_A to the binding sequence, therefore B^ρ achieves the operativeness of the party who sends the binding sequence, which is S^ρ. We are interested in the operativeness A^ρ, i.e., a party claiming to be A is executing A^ρ. The further analysis is as follows.

In order to send the binding sequence of B^ρ, S must have received a request of the form $\mathcal{E}_{K_{BS}}\big(A \| \mathcal{E}_{K_{AS}}(N_B)\big)$. Further, to send $\mathcal{E}_{K_{BS}}\big(A \| \mathcal{E}_{K_{AS}}(N_B)\big)$, the role program B^ρ must have received $\mathcal{E}_{K_{AS}}(N_B)$. Although B^ρ cannot read this message, the message $\mathcal{E}_{K_{AS}}(N_B)$ must be correct, otherwise the server program will not send $\mathcal{E}_{K_{AS}}(N_B)$. Therefore, we conclude that the party who claims to be A is currently present and therefore the operativeness FLAG for A^ρ is achieved.

Now we consider the willingness goal. The two willingness challenges are as follows.

$$\begin{aligned}\mathbf{C}_1 =& \mathcal{E}_{K_{BS}}(N_B) \quad \text{(Expected configuration: } A \longrightarrow B\text{)}\\ \mathbf{C}_0 =& \mathcal{E}_{K_{BS}}(N_B) \quad \text{(Alternate configuration: } A \longrightarrow C\text{)}\end{aligned}$$

Clearly, a distinguisher for the willingness goal cannot be constructed, because the two challenges are not different. We analyse whether $\mathbf{C}_0$ is inconsistent with any of the protocol assumptions. The challenge $\mathbf{C}_0$ implies that that B^ρ has sent a message for $\mathcal{E}_{K_{BS}}\big(A \| \mathcal{E}_{K_{AS}}(N_B)\big)$. Sending such a message implies that B previously received $\mathcal{E}_{K_{AS}}(N_B)$, which is possible even if A is authenticating itself to a party C, instead of B. Therefore, the role program B^ρ cannot conclude that A is necessarily willing to communicate with B. Thus, the willingness goal is not achieved.

6.2.2 Attacks on the WL protocol

The WL protocol is insecure if a system designer expects it to achieve the identification or willingness goal. Our analysis shows that the protocol is secure with respect to the operativeness goal.

Woo and Lam later describes two attacks on the protocol [158]. The first attack is a kind of typing attack and it does not work if the length of protocol terms are a priori known, as we assume in this thesis. The second attack, which is attributed to Martín Abadi, prevents the protocol from achieving the identification of A. This attack, however, does not prevent achieving the operativeness goal. The attack narration is as follows.

$$
\begin{array}{lll}
(1) & \mathcal{I}_1^\rho \to B_1^\rho\colon & A \\
(2') & \mathcal{I}_2^\rho \to B_2^\rho\colon & \mathcal{I}^\rho \\
(2) & B_1^\rho \to \mathcal{I}_1^\rho\colon & N_B \\
(2') & B_2^\rho \to \mathcal{I}_2^\rho\colon & N'_B \\
(3) & \mathcal{I}_1^\rho \to B_1^\rho\colon & rand \\
(3') & \mathcal{I}_2^\rho \to B_2^\rho\colon & \mathcal{E}_{K_{\mathcal{I}S}}(N_B) \\
(4) & B_1^\rho \to S^\rho\colon & \mathcal{E}_{K_{BS}}\big(A \| rand\big) \\
(4') & B_2^\rho \to S^\rho\colon & \mathcal{E}_{K_{BS}}\big(\mathcal{I} \| \mathcal{E}_{K_{AS}}(N_B)\big) \\
(5) & S^\rho \to B_1^\rho\colon & \mathcal{E}_{K_{BS}}(N_B)
\end{array}
$$

In this attack, an adversary executes two parallel sessions of the protocol, namely two attack programs $\mathcal{I}_1^\rho$ and $\mathcal{I}_2^\rho$ try to authenticate as A and $\mathcal{I}$ respectively, to an honest party B who correspondingly executes two copies of the role program B_1^ρ and B_2^ρ.

The party B sends two nonces N_B and N'_B to attack programs $\mathcal{I}_1^\rho$ and $\mathcal{I}_2^\rho$ respectively. At this point, in flow (3) and flow (3′), the two attack programs swap their nonces. Since $\mathcal{I}_1^\rho$ is pretending to be executed by A, it cannot encrypt N'_B with K_{AS}, and therefore it sends a random bit string to B. On the other hand, $\mathcal{I}_2^\rho$ encrypts N_B with $K_{\mathcal{I}S}$ and sends it back to B.

Since B is not aware of the fact that $\mathcal{I}_1^\rho$ and $\mathcal{I}_2^\rho$ have swapped their nonces, the program B_1^ρ sends $\mathcal{E}_{K_{BS}}\big(A \| rand\big)$ to the server assuming that rnd was sent by $\mathcal{I}_1^\rho$ that is being executed by A. In the parallel session, the program B_2^ρ sends $\mathcal{E}_{K_{BS}}\big(\mathcal{I} \| \mathcal{E}_{K_{AS}}(N_B)\big)$ to the server assuming that $\mathcal{E}_{K_{AS}}(N_B)$ was sent by $\mathcal{I}_2^\rho$ that is being executed by $\mathcal{I}$.

When B receives $\mathcal{E}_{K_{BS}}(N_B)$ from the server, B_1^ρ concludes that it is communicating with A while A is not present in the execution. Therefore, the identification

goal cannot be achieved by B. This was also indicated by our analysis, but this attack is an example scenario in which the identification goal is not achieved.

Even in the presence of the above attack, the operativeness goal is achieved, because the adversary cannot repeat the attack transcript. In every instance of the attack, the adversary must be operative while pretending to be A and compute a new value of $\mathcal{E}_{K_{IS}}(N_B)$.

It is trivial to construct an attack that only violates the willingness goal without violating the identification goal. For this purpose, an adversary proceeds as follows. When A sends his identity in order to authenticate itself to a party C, the adversary relays that message to another party B. Now, the nonce is received from B instead of C. The adversary sends this nonce to A, who then replies accordingly. There is no further communication of A with C, and the remaining session completes between C and the server S.

The WL protocol can be improved in several ways to achieve the identification and willingness goals. For example, Anderson and Needham [6] propose an improved version of the protocol:

$$
\begin{array}{lll}
(1) & A^\rho \rightarrow B^\rho: & M_1 = A \\
(2) & B^\rho \rightarrow A^\rho: & M_2 = N_B \\
(3) & A^\rho \rightarrow B^\rho: & M_3 = \mathcal{E}_{K_{AS}}(B \| N_B) \\
(4) & B^\rho \rightarrow S^\rho: & M_4 = A, \mathcal{E}_{K_{AS}}(B \| N_B) \\
(5) & S^\rho \rightarrow B^\rho: & M_5 = \mathcal{E}_{K_{BS}}(N_B \| A)
\end{array}
$$

There are two important difference between the improved protocol and the original WL protocol. First, the identity of A is included in M_5, which enables B to achieve the identification of A, by making the two identification challenges different. Second, the identity of B is included in M_3, which implies that A provides his willingness to communicate with B.

6.3 Summary

In this chapter, we demonstrated that a structured intuition based analysis provides more insight into the working of an authentication protocol. On one hand, an attack on a protocol does not necessarily mean that the protocol is completely insecure. On the other hand, a secure protocol is only secure with respect to a certain set of FLAGs, and if the protocol is used in an application where this set of FLAGs does not meet the application requirements then an adversary

may be able to construct an attack. One can also analyse a protocol in more detail, e.g., in one of our reports [5], we were able to find new vulnerabilities in Needham-Schroeder secret-key protocol [117] and Denning-Sacco protocol [58], in their CBC (cipher-block chaining mode) and CFB (cipher feedback mode) based implementations.

Chapter 7

Adaptable Security

A security model is traditionally based on a model of fixed and an all-powerful adversary. The use of an all-powerful adversary is motivated by a number of reasons. First, it is difficult to foresee potential applications of a protocol at the design time. The assumption of an all-powerful adversary makes security analysis applicable in a wide variety of applications. For example, the Dolev-Yao adversary [62] is believed to subsume all conceivable adversaries in most computer networks; this model is the most common choice in formal security models [105].

Second, the model of an all-powerful adversary is dictated by the inventor's paradox [129], namely it is often easier to analyse an abstract (and general) version of a security problem, which employs an abstract model of an adversary, as compared to analysing an application specific version of the problem and using an application specific model of an attacker. For example, it is easier to work with the assumption that an adversary can modify a message, as compared to the assumption that an adversary can change zeros to ones but he cannot change ones to zeros, such as in impulse transmission [11].

The main advantage of using an all-powerful adversary is that a security guarantee against an all-powerful adversary is also valid for any weaker adversary, but insecurity in the presence of an all-powerful adversary does not imply insecurity for a weaker adversary. The disadvantage of using an all-powerful ad-

versary model is that not all real-world applications require the highest level of security [92], because an application environment may limit the attacker's capabilities. On the other hand, some applications cannot afford to implement the required level of security due to resource constraints [43, 95]. These are all valid concerns for a system developer. Therefore, there exists a demand for a methodology that enables an application specific analysis or, what we call, an adaptable security analysis.

The SI methodology can be used as a framework for an adaptable analysis of authentication protocols, in which the values of different parameters of the authentication model are decided during an analysis. These different parameters include FLAGs, and assumptions about the adversary and application environment. In a sense, an adaptable security analysis of a protocol provides results that are closer to the actual level of security provided by the protocol in a given application environment.

Usually, entity authentication is just a part of complete correctness requirements of a protocol; other requirements may include key establishment for instance. But, there are many applications where only entity authentication is required. One such application domain, which we consider in this chapter, is RFID (Radio Frequency Identification). Sarma et al. [135] provide a good introduction to the RFID technology.

The design of an RFID authentication protocol is a challenging task, because RFID tags are resource constrained devices with only a little memory and limited computational and communication capabilities. These constraints mandate the use of light-weight cryptography, but this is not the only challenge. The Dolev-Yao attack model does not apply to RFID protocols, as RFID tags are not tamper-proof and therefore cannot be trusted. Moreover, privacy also needs to be considered in the design of RFID protocols. The use of RFID based identification in wearable items can lead to serious privacy concerns, e.g., some people may not want to be tracked by advertising companies if they wear clothes that have embedded RFID tags.

The SI methodology is useful for designing RFID authentication protocols, where it is not feasible to use a notion of security that does not take application constraints into account. The requirement of a trade-off between security and resources is essential for RFID based protocols. We, however, do not consider any of RFID domain specific problems, e.g., the efficiency of reader side of a protocol, side channel attacks and relay attacks. Solving these problems may also be crucial for the feasibility of an RFID based protocol.

7.1 Overview

The aim of a traditional security analysis is to show that a set of correctness goals can be achieved by executing an instance of a role program and assuming a certain class of adversaries. The examples of traditional analyses include reductionist type proofs of matching conversation [28, 31] and showing the authenticity of a message using a static program analysis [34].

An adaptable security analysis can be considered as solving an inverse problem, namely one tries to determine the conditions that can ensure that a protocol meets its correctness goals. In particular, given a set of FLAGs, a protocol, a loosely defined environment, and a generic attacker, a security analyst tries to determine the constraints on the environment and limits on the attacker's capabilities, such that the protocol achieves the set of FLAGs. For the SI methodology, an adaptable analysis corresponds to the following steps.

1. Hypothesise (or assume) a binding sequence of a role program.

2. Carry out the correctness analysis to validate a reasonable set of FLAGs from the binding sequence. If a FLAG cannot be achieved then repeat this step after introducing new assumptions regarding the instantiation of protocol messages and functions, trust assumptions, secret values, and the operating environment.

3. Carry out the dependency analysis and change functional relations so that the binding sequence is plausible.

4. Validate the binding sequence by carrying out a security analysis for the canonicity requirements of the binding sequence. If the security analysis fails then repeat this step by introducing appropriate assumptions about the adversary.

The first step is the process of hypothesising a binding sequence, without first carrying out the dependency and security analysis of the role program. The reason we hypothesise the binding sequence, instead of determining it after the security and dependency analysis, is that the security and dependency analysis will depend on the binding sequence in an adaptable analysis. We introduce assumptions and change dependencies between messages, in order to justify the binding sequence. Hypothesising a binding sequence is a trivial task, namely one can try all of the plausible binding sequences in the role program, because a typical role program receives a small number of messages.

The second step is the correctness analysis to derive FLAGs using the operational definitions of the FLAGs.
In the third step, we try to construct a dependency graph that spans all the messages of the binding sequence. In this step we may need to change functional relations between the messages in order to construct the dependency graph. If the dependency graph does not spans all the messages then the binding sequence cannot be justified, and we will need to change our hypothesis of the first step.

In the last step, we carry out the security analysis, in which certain messages of the binding sequence are shown to be canonical messages. The requirements of canonicity depend on the shape of the dependency graph. We may need to introduce additional assumptions, in order to weaken the adversarial model, so that the canonicity requirements can be met.

7.2 RFID System

We present a system model that represents the use of RFID tags in a typical retail shop. We consider low cost RFID tags, namely passive transponders that have a limited amount of memory and computational capability. There is one reader R in the system. The assumption of a single reader is ubiquitous in most of the RFID literature [154, 55].

The tags are attached to shelf items in a retail shop. The identity of a tag is Id, which is mapped to the name of the item to which the tag is attached. This identity is not necessarily unique for each RFID tag. There are n different tag identities: $1 \leq Id \leq n$. The tag identities are known values in the system.

There is a single entry point *Entry* into the shop and a single exit point *Exit* from the shop. Whenever an item passes through *Entry*, it gets attached with a tag Id and a database entry $(Id, K, \text{information})$ is stored in the reader's database db:

$$db = \{(Id, K, .) : 1 \leq Id \leq n\}$$

This database entry includes a field K, which is a cryptographic key and is stored on the tag. The identity Id of the tag is not stored on the tag. The database entry may also include additional information, e.g., retail price and discount policy. We assume that keys are generated from the uniform distribution. We assume that the size of a key, $|K|$, is sufficiently long so that the probability

that a randomly generated key corresponds to a database entry is negligible in $|K|$.

A customer comes to the shop and places items in his basket. For check-out, he arrives at *Exit*. The reader that is installed at *Exit* interrogate the tags of purchased items by executing an authentication protocol, in order to retrieve the database entries of the tags. The retrieved information is used to find the pricing information and generate an invoice for the customer.

The reader R may have multiple front-ends at *Exit* (corresponding to multiple checkout terminals), but the reader has one common back-end for the database. In reality, the back-end database can be a distributed storage if it is synchronized in real-time. Whenever a tag is in *Exit*, it is in the reader range and an authentication protocol is automatically executed.

RFID Protocol

We consider a generic RFID protocol Π. The generic protocol consists of two role programs: $\rho_1(.)$, which is executed by the reader, and $\rho_2(.)$, which is executed by an RFID tag.

$$\begin{aligned} \text{(Interrogation)} \quad & \rho_1(.) \longrightarrow \rho_2(.) : M_1 = V_R \\ \text{(Response)} \quad & \rho_2(.) \longrightarrow \rho_1(.) : M_2 = F_K(V_R, V_{Id}) \end{aligned}$$

Many existing protocols follow this generic form, e.g., weak-private RFID schemes [154]. In this generic protocol, M_1 is a reader challenge while M_2 is a tag response. The function $F_K(...)$ represents either a keyed hash functions, or a lightweight block cipher. If unstated, $F_K(...)$ stands for a keyed hash function. The term V_R contains a value generated by the reader and the term V_{Id} contains a value generated by a tag. The reader R can find the entry $(Id, K, .)$ in the database by searching the value of K in the database, such that the predicate $M_2 = F_K(V_R, .)$ is true. Since K is not a part of the response, the reader R must be able to search the whole database to find the correct entry, which takes a linear amount of time in the size of the database.

We construct eight different concrete protocols based on the following three parameters of the generic protocol:

1. Whether the challenge V_R is random or fixed: If the challenge is random then V_R is a variable, and if the challenge is fixed then $V_R = a$, where a is a constant.

	Interrogation	Response
Π^1	a	$F_k(a, b)$
Π^2	V_R	$F_k(V_R, b)$
Π^3	a	$F_k(a, V_{Id})$
Π^4	V_R	$F_k(V_R, V_{Id})$
Π^5	a	$F_K(a, b)$
Π^6	V_R	$F_K(V_R, b)$
Π^7	a	$F_K(a, V_{Id})$
Π^8	V_R	$F_K(V_R, V_{Id})$

Table 7.1: Eight RFID protocols

2. Whether the value V_{Id} is random or fixed: If V_{Id} is considered a random value then V_{Id} stands for a variable, and if V_{Id} is considered fixed then $V_R = b$, where b is a constant.
3. Whether the key K is different for each tag or not: If the key is random then K stands for a variable, and if the key is fixed then $K = k$, where k is a constant.

With different values of these three parameters, we get eight protocols, which are denoted by $\Pi^1, \Pi^1, ..., \Pi^8$, which are listed in Table 7.1.

Correctness Requirements

As usual, correctness requirements are system dependent. In our case, correctness means a set of privacy and entity authentication requirements.

We define privacy as protection of the relation between a purchased item (which has an embedded RFID tag) and the customer. This notion of privacy is aimed at avoiding profiling and tracking of customers at a large scale. We assume that the adversary is not capable of "physically observing" an item all the way to a customer. In such cases, it seems difficult to ensure privacy, e.g., if a spy is following his target. For privacy, we are only interested in those adversaries that aim to automate this process at a large scale, e.g., using a large number of RFID readers in a shopping mall.

If an adversary can create a link between responses of a tag then he can track an RFID tag in a certain area. This means he can effectively track a person who carries the tag.

Definition 7.1 (Privacy requirement) A protocol consisting of an initiator program $\rho_1(.)$ executed by the reader R, and a responder program $\rho_2(.)$ executed by an RFID tag Id, protects privacy of the tag if an adversary that executes $\rho_1(.)$ cannot achieve the recognition of the tag, i.e., $Recog((R \rhd Id, .)$ cannot be achieved.

If a tag can be recognized then the adversary can track the tag by repeated interrogations, which will violate the privacy of the person who carries that tag. The requirements for entity authentication are as follows.

Definition 7.2 (Authentication requirements) The reader role program $\rho_1(.)$ while communicating with a tag's role program $\rho_2(.)$ achieves authentication of the tag if the identification goal, $Idnt(R \rhd Id, \beta_1)$, and the operativeness goal $Oper(R \rhd Id, \beta_1)$, are achieved by $\rho_1(.)$.

Entity authentication can be interpreted in different ways depending on the application. So, Def. 7.2 is just one definition of entity authentication, which is used in this chapter. The two FLAGs in the definition are sufficient for the system that is considered here, and for example there is no need for mutual authentication between an RFID reader and an RFID tag.

The eight protocols, specified in Table 7.1, are analysed for the correctness requirements using the SI methodology. Recall that there are four steps involved in an adaptable authentication model. The first step consists of forming a hypothesis for a binding sequence. This step is trivial for the generic protocol, namely $\beta_1 = F_K(V_R, V_{Id})$ is the generic binding sequence for the reader program. The second step is the correctness analysis, which is described in the next section.

7.3 Correctness Analysis

For the authentication of a tag, we corroborate the operational definitions of the two FLAGs: $Idnt(R \rhd Id, \beta_1)$ and $Oper(R \rhd Id, \beta_1)$. For the privacy requirement, we corroborate that $Recog((\mathcal{I} \rhd Id, \beta_1)$ cannot be achieved by an adversary $\mathcal{I}$. It is not possible to achieve the correctness requirements for each protocol without any additional assumptions, therefore, we also introduce assumptions during the correctness analysis.

A few notational conventions are as follows. The values of terms in an r-th run are denoted by $N_{Id}(r)$ and $N_R(r)$. Let $s = |F_K(...)|$ and $|N_R| = |N_{Id}|$. Let n be

total number of tags in the system, such that $n < n_{th}$, for a suitable choice of the threshold n_{th}. Let q be the total number of queries to $F_K(...)$ and $q < q_{th}$, for a suitable value of the threshold q_{th}.[1] We rely on an asymptotic analysis for calculating different probabilities. This means that the probabilities, $q.2^{-s}$, $q.2^{-|N_R|}$, $q.2^{-|N_{Id}|}$ and $n.2^{-|K|}$ are assumed to be negligible. The correctness analysis of the eight protocols is as follows.

Π^1: Fixed key, Fixed challenge, Deterministic response

The narration of Π^1 is as follows, in which a, b, and k are fixed values.

$$\begin{array}{ll} \rho_1(.) \longrightarrow \rho_2(.): & M_1 = a \\ \rho_2(.) \longrightarrow \rho_1(.): & M_2 = F_k(a,b) \end{array}$$

As per the operational definition of $Idnt(R \triangleright Id, \beta_1)$, the reader program $\rho_1(\bar{k}, \bar{a})$ should be able to distinguish between $\beta_1(r) = F_k(a,b)$ (for a tag Id) and $\beta_1(r') = F_k(a,b)$ (for another tag $Id' \neq Id$). As $\beta_1(r) = \beta_1(r')$, the distinguishing is not possible. This means that the identification goal cannot be achieved by the reader's role program. Therefore, we need additional assumptions that ensure that the reader does not depend on the protocol Π^1 for the identification goal. We introduce two assumptions for Π^1, which are as follows:

Assumption (1): The shop contains the same type of items, or there is an auxiliary (e.g., physical) mechanism to distinguish between different types of items.
Assumption (2): A single item is presented to the reader R at a time.

To achieve $Oper(R \triangleright Id, \beta_1)$, there must be a dependency arc from a nonce of $\rho_1(.)$ to the binding sequence of $\rho_1(.)$. Clearly, this is not possible because there is no nonce in $\rho_1(.)$ of Π^1. So, we include an assumption that ensures that the reader does not rely on the operativeness goal.

Assumption (3): Visual inspection should be carried out at *Exit* to make sure the item with the tag Id is present there.

For the privacy, an adversary should not be able to achieve the recognition goal. As per the operational definition of recognition on Pg. 98, it is required that an adversary should not be able to pick a pair of binding sequences with a probability different from the rest of two pairs. Since all binding sequences of $\rho_1(.)$

[1] For instance, q_{th} may represent an upper bound of the birthday paradox.

are the same, we conclude that the adversary cannot achieve the recognition of a tag and the protocol meets the requirement of privacy.

Π^2: Fixed key, Random challenge, Deterministic response

The narration of Π^2 is as follows.

$$\begin{array}{ll} \rho_1(.) \longrightarrow \rho_2(.): & M_1 = V_R \\ \rho_2(.) \longrightarrow \rho_1(.): & M_2 = F_k(V_R, b) \end{array}$$

For $Idnt(R \rhd Id, \beta_1)$, the reader program $\rho_1(\bar{k}, \bar{N}_R)$ should be able to distinguish between $\beta_1(r) = F_k(N_R(r), b)$ (for a tag Id) and $\beta_1(r') = F_k(N_R(r'), b)$ (for another tag Id'). Such a distinguishing is not possible as response is of the same type for every tag. So, we include assumption (1) and assumption (2).

To achieve $Oper(R \rhd Id, \beta_1)$, there must be a dependency arc from a nonce of $\rho_1(.)$ to the binding sequence of $\rho_1(.)$. In Π^2, there is a dependency arc from N_R to $F_k(N_R, b)$. There are two error events associated with this arc. First, if two random numbers are equal: $N_R(r) = N_R(r')$. Second, if there is a collision in the hash function resulting in $F_k(N_R(r), b) = F_k(N_R(r'), b)$. Since N_R is the output of a pseudo-random generator, the first event can only happen with negligible probability, $q.2^{-|N_R|}$. Since $F_k(.)$ is a collision resistance function, the second event also occurs with a negligible probability, $q.2^{-s}$. So, we conclude that $Oper(R \rhd Id, \beta_1)$ is achieved.

For the recognition, the three binding sequences are as follow: $\beta_1(r) = F_k(N_R(r), b)$ (between R and Id), $\beta_1(r') = F_k(N_R(r'), b)$ (between R and Id) and $\beta_1(r'') = F_k(N_R(r''), b)$ (between R and Id'). For the privacy, it is required that an adversary should not be able to pick a pair of binding sequences with a probability different from the rest of two pairs. Since all the tags have the same key k, there is no difference between the binding sequences of different tags. Thus, the protocol Π^2 meets the privacy requirement.

Π^3: Fixed key, Fixed challenge, Randomised response

The narration of Π^3 is as follows.

$$\begin{array}{ll} \rho_1(.) \longrightarrow \rho_2(.): & M_1 = a \\ \rho_2(.) \longrightarrow \rho_1(.): & M_2 = F_k(a, V_{Id}) \end{array}$$

For the identification, the reader program $\rho_1(\bar{k}, \bar{a})$ should be able to distinguish between $\beta_1(r) = F_k(a, N_{Id}(r))$ and $\beta_1(r') = F_k(a, N_{Id}(r'))$. This is not possible, because $N_{Id}(r)$ and $N_{Id}(r')$ are randomly generated by the tags. So, we include assumption (1) and assumption (2).

To achieve $Oper(R \rhd Id, \beta_1)$, there must be a dependency arc from a nonce of $\rho_1(.)$ to the binding sequence of $\rho_1(.)$. Clearly, this is not possible because there is no nonce in $\rho_1(.)$ of Π^3. So, we include assumption (3). Since $\rho_1(\bar{k}, \bar{a})$ does not know the value of N_{Id} and by default $F_k(...)$ is a keyed hash function, we also include assumption (4).

Assumption (4): The function $F_k(...)$ is a PRP (pseudo-random permutation) with an efficiently computable inverse function $F_K^{-1}(...)$, e.g., a block cipher [35].

For the recognition, the three binding sequences are as follow: $\beta_1(r) = F_k(a, N_{Id}(r))$ (between R and Id), $\beta_1(r') = F_k(a, N_{Id}(r'))$ (between R and Id) and $\beta_1(r'') = F_k(a, N_{Id}(r''))$ (between R and Id'). Since N_{Id} is assigned a random value, the adversary cannot achieve the recognition of tags. Thus, the protocol Π^3 achieves the privacy goal.

Π^4: Fixed key, Random challenge, Randomised response

The narration of Π^4 is as follows.

$$\begin{array}{ll} \rho_1(.) \longrightarrow \rho_2(.): & M_1 = V_R \\ \rho_2(.) \longrightarrow \rho_1(.): & M_2 = F_k(V_R, V_{Id}) \end{array}$$

As per operational definition of identification, the reader program $\rho_1(\bar{k}, \bar{N}_R)$ should be able to distinguish between $\beta_1(r) = F_k(N_R(r), N_{Id}(r))$ and $\beta_1(r') = F_k(N_R(r'), N_{Id}(r'))$. Similar to Π^1, Π^2 and Π^3, this is not possible as the key is fixed. We include assumption (1) and assumption (2). Since, $\rho_1(\bar{k}, \bar{N}_R)$ does not know the value of N_{Id}, so we further include assumption (4): $F_k(...)$ is a PRP with an efficiently computable inverse function $F_k^{-1}(...)$.

To achieve $Oper(R \rhd Id, \beta_1)$, there must be a dependency arc from a nonce of $\rho_1(.)$ to the binding sequence of $\rho_1(.)$. In Π^4, there is a dependency arc from N_R to $F_k(N_R, N_{Id})$, which is defined by a non-deterministic dependency function, because N_{Id} is not known to R. Due to assumption (4), the dependency function is verifiable by the corresponding decryption function. There is an error event

associated with the dependency arc, namely if in two different runs the values of N_R are the same. Since N_R is the output of a pseudo-random generator, the event can only happen with negligible probability, $q.2^{-|N_R|}$. So, we conclude that $Oper(R \triangleright Id, \beta_1)$ is achieved.

The three binding sequences are as follows: $\beta_1(r) = F_k(N_R(r), N_{Id}(r))$ (between R and Id), $\beta_1(r') = F_k(N_R(r'), N_{Id}(r'))$ (between R and Id) and $\beta_1(r'') = F_k(N_R(r''), N_{Id}(r''))$ (between R and Id'). Since adversary does not know N_{Id}, the distinguishing is not possible. Thus, the protocol Π^4 achieves the privacy goal.

Π^5: Random key, Fixed challenge, Deterministic response

The narration of Π^5 is as follows.

$$\begin{aligned} \rho_1(.) &\longrightarrow \rho_2(.): \quad M_1 = a \\ \rho_2(.) &\longrightarrow \rho_1(.): \quad M_2 = F_K(a, b) \end{aligned}$$

For $Idnt(R \triangleright Id, \beta_1)$, the reader program $\rho_1(\bar{db}, a)$ should be able to distinguish between two identification challenges: $\mathbf{C}_1 = \beta_1(r) = F_{K_{Id}}(a, b)$ (between R and Id); $\mathbf{C}_0 = \beta_1(r') = F_{K_{Id'}}(a, b)$ (between R and Id'). Here, K_{Id} and $K_{Id'}$ are the keys shared with Id and Id' respectively by the reader R. The distinguishing is possible as the stored keys are different for every tag. For a tag Id, the entry $(Id, K, .)$ can be retrieved from the database to compute the identity of tag.

To achieve $Oper(R \triangleright Id, \beta_1)$, there must be a dependency arc from a nonce of $\rho_1(.)$ to the binding sequence of $\rho_1(.)$. Clearly, this is not possible because there is no nonce in $\rho_1(.)$ of Π^5. So, we include assumption (3).

For the privacy goal, the three binding sequences are as follows: $\beta_1(r) = F_{K_{Id}}(a, b)$ (between R and Id); $\beta_1(r') = F_{K_{Id}}(a, b)$ (between R and Id); $\beta_1(r'') = F_{K_{Id'}}(a, b)$ (between R and Id'). The first two binding sequences are exactly same and different from the third one. An adversary can easily recognize the pair $(\beta_1(r), \beta_1(r))$ as different from the other two pairs. Clearly, the privacy of tags is not preserved, and, e.g., an adversary can link the replies from a tag to track the tag outside the shop. Therefore, we include the following assumption (5).

> Assumption 5: When a tag passes through the *Exit*, it is killed by the reader, so that no further communication is possible with the tag; alternatively, the tag is physically removed from the item.

Π^6: Random key, Random challenge, Deterministic response

The narration of Π^6 is as follows.

$$\begin{array}{ll} \rho_1(.) \longrightarrow \rho_2(.): & M_1 = V_R \\ \rho_2(.) \longrightarrow \rho_1(.): & M_2 = F_K(V_R, b) \end{array}$$

For $Idnt(R \triangleright Id, \beta_1)$, the reader program $\rho_1(\bar{db}, \bar{N}_R)$ should be able to distinguish between $\beta_1(r) = F_{K_{Id}}(N_R(r), b)$ and $\beta_1(r') = F_{K_{Id'}}(N_R(r'), b)$. Clearly, such a distinguishing is possible as a different key is used for each tag.

To achieve $Oper(R \triangleright Id, \beta_1)$, there must be a dependency arc from a nonce of $\rho_1(.)$ to the binding sequence of $\rho_1(.)$. In Π^6, there is a dependency arc from N_R to $F_K(N_R, b)$. There are two error events associated with this arc. First, if two random numbers are equal: $N_R(r) = N_R(r')$. Second, if there is a collision in the hash function resulting in $F_k(N_R(r), b) = F_k(N_R(r'), b)$. Since N_R is the output of a pseudo-random generator, the first event can only happen with negligible probability, $q.2^{-|N_R|}$. Since $F_k(.)$ is a collision resistance function, the second event also occurs with a negligible probability of $q.2^{-s}$. So, we conclude that $Oper(R \triangleright Id, \beta_1)$ is achieved.

For the privacy goal, the three binding sequences are as follow: $\beta_1(r) = F_{K_{Id}}(N_R(r),$ $b)$ (between R and Id); $\beta_1(r') = F_{K_{Id}}(N_R(r'), b)$ (between R and Id); $\beta_1(r'') = F_{K_{Id'}}(N_R(r''), b)$ (between R and Id'). The privacy of tags is not preserved, because the third binding sequence uses a different key. An adversary can repeatedly interrogate a tag with a fixed value of N_R to track it outside the shop. Therefore, we include assumption (5).

Π^7: Random key, Fixed challenge, Randomised response

The narration of Π^7 is as follows.

$$\begin{array}{ll} \rho_1(.) \longrightarrow \rho_2(.): & M_1 = a \\ \rho_2(.) \longrightarrow \rho_1(.): & M_2 = F_K(a, V_{Id}) \end{array}$$

For $Idnt(R \triangleright Id, \beta_1)$, the reader program $\rho_1(\bar{db}, a)$ should be able to distinguish between $\beta_1(r) = F_{K_{Id}}(a, N_{Id}(r))$ and $\beta_1(r') = F_{K_{Id'}}(a, N_{Id}(r'))$. In principle, such a distinguishing is possible as K is unique for each tag, but there is another problem: $\rho_1(\bar{db}, a)$ does not know the value of N_{Id}. Therefore, we include

assumption (4): $F_K(...)$ is a PRP with an efficiently computable inverse function $F_K^{-1}(...)$.

To achieve $Oper(R \rhd Id, \beta_1)$, there must be a dependency arc from a nonce of $\rho_1(.)$ to the binding sequence of $\rho_1(.)$. Clearly, this is not possible because there is no nonce in $\rho_1(.)$ of Π^7. So, we include assumption (3).

For the privacy goal, the three binding sequences are as follow: $\beta_1(r) = F_{K_{Id}}(a, N_{Id}(r))$ (between R and Id); $\beta_1(r') = F_{K_{Id}}(a, N_{Id}(r'))$ (between R and Id); $\beta_1(r'') = F_{K_{Id'}}(a, N_{Id}(r''))$ (between R and Id'). The replies from a tag are randomized by N_{Id}, therefore recognition of the tags is not possible and the privacy goal is achieved.

Π^8: Random key, Random challenge, Randomised response

The narration of Π^8 is as follows.

$$\begin{aligned} \rho_1(.) &\longrightarrow \rho_2(.): \quad M_1 = V_R \\ \rho_2(.) &\longrightarrow \rho_1(.): \quad M_2 = F_K(V_R, V_{Id}) \end{aligned}$$

For $Idnt(R \rhd Id, \beta_1)$, the reader program $\rho_1(\bar{db}, \bar{N}_R)$ should be able to distinguish between $\beta_1(r) = F_{K_{Id}}(N_R(r), N_{Id}(r))$ and $\beta_1(r') = F_{K_{Id'}}(N_R(r'), N_{Id}(r'))$. Such a distinguishing is possible as K is unique for each tag. Again, $\rho_1(\bar{db}, \bar{N}_R)$ does not know the random value of N_{Id}, therefore, we include assumption (4).

To achieve $Oper(R \rhd Id, \beta_1)$, there must be a dependency arc from a nonce of $\rho_1(.)$ to the binding sequence of $\rho_1(.)$. In Π^8, there is a dependency arc from N_R to $F_K(N_R, N_{Id})$, which is defined by a non-deterministic dependency function, because N_{Id} is not known to R. Due to assumption (4), the dependency function is verifiable by the corresponding decryption function. There is an error event associated with the dependency arc, corresponding to the case when, in two different runs, the values of N_R are the same. Since N_R is the output of a pseudo-random generator, the event can only happen with negligible probability, $q.2^{-|N_R|}$. So, we conclude that $Oper(R \rhd Id, \beta_1)$ is achieved.

For the privacy goal, the three binding sequences are as follow: $\beta_1(r) = F_{K_{Id}}(N_R(r), N_{Id}(r))$ (between R and Id); $\beta_1(r') = F_{K_{Id}}(N_R(r'), N_{Id}(r'))$ (between R and Id); $\beta_1(r'') = F_{K_{Id'}}(N_R(r''), N_{Id}(r''))$ (between R and Id'). The replies from a tag are randomized by N_{Id}, therefore the privacy goal is achieved.

This completes our correctness analysis. The next step of the adaptable analysis is the dependency analysis, which is trivial for these eight protocols, because there is only one dependency node corresponding to the second message of the generic protocol. The last step of the adaptable analysis is the security analysis, which is described in the next section.

7.4 Security Analysis

The binding sequence of each of the eight protocols consists of just one message variable M_2 (the reply from a tag). Therefore, in the security analysis we investigate that whether M_2 is always assigned with a canonical message. This essentially means that the value of M_2 should be from a run executed by a legitimate tag. The adversary can replay a canonical message.

7.4.1 Adversary classes

We present an adversarial model that is relevant to RFID based systems and in particular to the environment of a typical retail shop. We assume that an adversary is computationally bounded, and asymptotic security arguments apply to him, e.g., if the key size $|K|$ is sufficiently large then the adversary cannot break an encryption function that uses that key.

The adversary interacts with its environment using a set of oracles. Each oracle represent a capability of the adversary. The adversary exists inside the shop as well as outside of the shop, and he can invoke the oracle queries following the rules of its class, which are defined later.
There are four basic oracles that are available to all adversaries.

- $CreateTag(K)$: An adversary can re-program a tag with a new key K. The adversary cannot read the tag memory with this oracle.
- $Launch(Id, M_1)$: An adversary acts as a reader and interacts with the tag Id using the challenge M_1 of his choice. The adversary is inside the shop, and he selects the tag Id for interaction.
- $Respond(M_2)$: An adversary acts as a tag Id and responds to a reader's interrogation with M_2 as the response.
- $Ping(M_1)$: The adversary can interrogate RFID tags outside the shop with a challenge M_1 to collect the responses of the tags that are present

in the interrogation range. This oracle can be used to track a customer outside the shop.

The following two oracles are available to the adversary depending on his class.

- $Corrupt(Id)$: An adversary tampers with the tag Id to read the key stored in it. We assume that a tampered tag is detectable and the corresponding entry is removed from the database.
- $Admin(Id)$: An adversary can read the memory of the tag Id without destroying it. This oracle models an insider adversary who is one of the system administrators and knows the values of keys stored in all tags. An insider adversary, however, cannot compromise the integrity of the database, e.g., he cannot insert fake entries in the database.

We define three classes of adversaries.

Insider Class $\mathbf{I}^A$: An adversary $\mathcal{I} \in \mathbf{I}^A$ can access all of the oracles. Note that with the access to $Admin(Id)$ there is no need to rely on $Corrupt(Id)$, which destroys the tag Id.

Destructive class $\mathbf{I}^D$: An adversary $\mathcal{I} \in \mathbf{I}^D$ (Destructive class) can access all of the oracles except $Admin(Id)$. If $\mathcal{I}$ uses the oracle $Corrupt(Id)$ then the tag Id is destroyed.

Weak class $\mathbf{I}^W$: An adversary $\mathcal{I} \in \mathbf{I}^W$ cannot access $Corrupt(Id)$ and $Admin(Id)$ oracles.

Clearly, the three adversary classes are related: $\mathbf{I}^W \subset \mathbf{I}^D \subset \mathbf{I}^A$. Note that for the insider class there is no secret value in the system, because an insider adversary knows all of the keys stored in RFID tags.

7.4.2 Canonicity Analysis

We note that $CreateTag(K)$ oracle is not relevant to the canonicity analysis, because the database cannot be updated to include the fake entry corresponding to K. Similarly, $Ping(M_1)$ oracle is also not relevant to the canonicity analysis, because this oracle is used outside of the shop; the canonicity requirement is

applied at the *Exit*, where the honest reader R interrogate an RFID tag. Inside of the shop, an adversary uses $Launch(M_1, Id)$ oracle to interrogate RFID tags.

In an execution of the protocol Π_1, a message for M_2 is always canonical in presence of an insider adversary. The canonical message in this case is a known constant $F_k(a, b)$. If the adversary plays any message other than $F_k(a, b)$ it is not accepted by the reader. The fixed key k, which is used in the protocol, does not need to be secret. Therefore, $Admin()$ and $Corrupt(Id)$ are not useful to the adversary.

In an execution of Π_2, the message for M_2 is canonical in presence of a weak adversary. An insider or destructive adversary can learn the fixed key k easily, and then he can reply to a reader interrogation using $Respond(M_2)$ oracle. If the key k remains secret then the adversary cannot generate a valid response to an interrogation consisting of a random challenge N_R. The same arguments apply to Π_3 and Π_4.

For the last four protocols Π_5, Π_6, Π_7, and Π_8 each tag gets a different key K and is uniquely identifiable by K. This means that responses from different tags are different. If the key of a tag is not secret then an adversary can emulate the tag. This means that the canonicity of M_2 cannot guaranteed against an insider adversary.

The canonicity of M_2 holds for a destructive adversary in the last four protocols. This is because the oracle $Corrupt(Id)$ destroys the tag Id and the corresponding key cannot be re-used. Since every tag has a different key, the key of a corrupted tag cannot be used to learn the keys of other tags. This means the oracle $Corrupt(Id)$ is not useful.

7.5 Summary

The results of the analysis are summarized in Table 7.2. Each row in the table corresponds to one of the concrete protocols; the specific choices made for V_R, V_{Id} and K are mentioned in the corresponding columns. The last column lists the assumptions that are required to justify the security and privacy of these protocols.

Let us consider, for instance, the case Π^2 in the table. The protocol Π^2 corresponds to a system where all RFID tags share a common key and there is no pseudo-random generator implemented on the tags. The r-th interrogation of the reader consists of a random challenge $N_R(r)$. As shown in the last column

Protocol	K is random	V_{ID} is random	V_R is random	Results
Π^1	No	No	No	1,2,3,Insider
Π^2	No	No	Yes	1,2,Weak
Π^3	No	Yes	No	1,2,3,4,Weak
Π^4	No	Yes	Yes	1,2,4,Weak
Π^5	Yes	No	No	3,5,Destructive
Π^6	Yes	No	Yes	5,Destructive
Π^7	Yes	Yes	No	3,4,Destructive
Π^8	Yes	Yes	Yes	4,Destructive

Table 7.2: Concrete Forms of the Generic Protocol

that the protocol is correct against Weak class of adversaries, as long as the assumption, (1) and (2), are satisfied. This illustrates the types of results that we obtain in adaptable security analysis, namely appropriate parameters of an authentication model that are required to justify a set of correctness requirements.

In our adaptable model, we consider authentication and privacy goals as assumptions, even for the weakest protocol Π^1. We infer an adversary model and a set of assumptions about the environment such that the correctness requirements are justified at a system level. It is not always possible to justify every correctness requirement in an adaptable security model. Since our correctness goals are formulated at primitive level (compared to a single high level formulation, e.g., matching conversation [28]), not able to achieve a FLAG does not mean that other FLAGs cannot be achieved. A subset of original goals may be achievable.

In a traditional security analysis, most of the protocols in Table 7.2 are insecure, with the exceptions of Π^6 and Π^8. This is due to a strict deductive style interpretation of a security argument and the use of a fixed adversarial model. For example, if V_R is not random in a concrete protocol (i.e., one of Π^1, Π^3, Π^5, and Π^7) then the protocol is insecure under a Dolev-Yao adversary [62], because the protocol is prone to replay attacks.

Chapter 8

Related Work

There is a large body of work related to authentication protocols. We describe the most relevant work from two perspectives. First, we describe existing definitions of entity authentication and compare them to our FLAGs. Second, we briefly describe the contemporary techniques of protocol analyses.

8.1 Definitions

Characterization of security properties is a difficult task, and this is especially true for entity authentication. In most cases, a formal definition depends on the method used for the protocol analysis. Gollmann [78] provides a detailed analysis of the subject. He argues that formal definitions, although they add precision, do not often clarify the concept of entity authentication. Focardi et al. [70] also highlights the same issue that formalizing authentication goals is an error prone task. Even when a formal definition is given, it is hard to compare to others formalisms due to different mathematical assumptions. In our view, the main reason for the characterization problem is due to the lack of demarcation between service oriented goals and intermediate protocol dependent requirements.

Standard Definition

The international standard ISO/IEC 9798 specifies a number of authentication protocols. The standard consists of six parts. The first part [143] specifies general requirements and constraints of authentication mechanisms. The definition of entity authentication that appears in the first part was discussed in Chapter 1, which essentially states two requirements for entity authentication: identification and operativeness. Menezes et al. [106] provide a clearer definition than the standard definition, but their definition is also not satisfactory, as demonstrated in an attack on a protocol that meets the Menezes's definition [40].

Gollmann [79] also raises some concerns about the standard definition, in particular, regarding the level of abstraction, and meaning of an identity and claim. Based on his observations, he defines three goals related to entity authentication, which are referred to as G2, G3, and G4. (G1 is related to key establishment.) The first authentication goal G2 states that if a party receives a message that shows that the cryptographic key of a peer entity is used then authentication of the peer entity is achieved. Clearly, this goal captures the requirement of entity identification. The second goal G3 requires that a protocol run should be defined by the nonce of a party who is authenticating a far-end party. This is comparable to our notion of operativeness. The third goal G4 requires that the origin of all messages of a protocol must be authenticated. It is not clear which aspect of entity authentication is covered by this goal, as also indicated by the author himself. These definitions are protocol dependent, because they make use of private keys and sessions keys. Our definitions of FLAGs are abstract and are independent of protocol details.

Matching Conversations

The first formal definition of entity authentication in a cryptographic model appears in 1994 [28], which is proposed by Bellare and Rogaway and is called matching conversations. This definition is a refinement of its informal version proposed by Diffie et al. [59], which is called matching histories and which requires protocol parties to have the same history of messages in their respective runs.

A matching conversation is achieved by a protocol if an adversary can only faithfully relays messages between the protocol parties. For a two-role protocol, this requirement means that protocol messages are correctly interleaved and there is a one-to-one correspondence between the two transcripts. An entity authentication protocol is considered secure if protocol parties only accept those

runs that have matching conversations.

This definition is used in other work aiming at provable security of authentication protocols [31]. In our view, this model is too strong for entity authentication, because not all messages in a protocol necessarily contribute towards entity authentication. Further, it is unclear how one can interpret service oriented goals (such as FLAGs) from matching conversations, and whether correct interleaving is indeed necessary for entity authentication.

Correspondence

Woo and Lam [157] introduce the idea of correspondence assertions to specify authentication. A correspondence assertion stands for the requirement that if an event occurs in a run then a certain (corresponding) event must have already occurred. In their method, each message is annotated with labels indicating the progress of the protocol for each entity. A label could be of type begin-event or end-event, and it also includes the name of the party. A protocol is considered correct if in all protocol runs and in the presence of an adversary, every end-event label corresponds to a unique and earlier begin-event labelled with the same name. This requirement also does not capture service oriented authentication goals, and no method was provided to derive these goals from correspondence assertions. This definition has been used in many other works, most notably by Gordon and Jeffrey in Spi-calculus [82], and by Blanchet [33].

Intensional vs. Extensional Requirements

Roscoe [133] proposes intensional style of specifications, in which the goals of authentication are specified as patterns of messages as anticipated by the protocol designer. Roscoe proposes a canonical intensional requirement, which states that a run is successful only if a correct series of messages occur up to and including the last message. A matching conversation [28] is another example of an intensional style specification.

It might not be clear how an intensional property is related to the intuitively understandable goals of authentication, which are defined at a more abstract level and are called extensional goals by Roscoe. As Boyd and Mathuria [40] indicate, intensional specifications can be restrictive in a sense that an attacker may easily violate intensional specifications without affecting any of the corresponding extensional goals. The intensional style is not intuitive for protocol users. Gollmann [78] goes one step further and even discourages the use of such

formal specifications in general, by arguing that they create more confusion about the actual meaning of authentication, in particular, an intensional styled specification does not tell us what a protocol does.

Agreement

Lowe introduced an hierarchy of authentication goals [97] in 1997. He defines four basic goals, which can be interpreted with and without the notion of time. Arguably, this hierarchy is the most popular definition of entity authentication in formal security models. Unfortunately, FLAGs and Lowe's authentication definitions are not formally comparable. In the following, we present Lowe's definitions using our notations.

The weakest form of authentication is called aliveness. A party A with a protocol Π achieves the aliveness goal for a party B if B once executed a run of a role program of Π. Note that for aliveness it is not important that B has executed a correct role program, e.g., aliveness can still be achieved if both A and B have executed the same role program.

From correctness point of view, aliveness is comparable to our notion of identification. From security point of view, however, aliveness and canonicity are not comparable. Aliveness does not require that B is running the role program as expected by A, while for canonicity it is essential that B is running the expected program. On the other hand, canonicity does not require that it must be B who is running the expected role program; any network party can execute the expected role program. For aliveness, the identity of B is important.

At the correctness level, identification is a stronger requirement than aliveness, because the identification goal is derived from a list of canonical messages, which means that achieving identification involves two tasks: finding the identity (in the correctness analysis) and the execution of an expected role program (in the security analysis). This comparison is somewhat unfair, since aliveness is a security requirement that needs to be validated in presence of a network adversary. Security analysis is usually harder than (non-security) correctness analysis.

Lowe's second authentication goal is called weak agreement, which implies the following two requirements:

1. A party A achieves aliveness goal for a party B.
2. The run on B, in which the aliveness is achieved, must be with A.

At a conceptual level, this is comparable to the case when A achieves both identification and willingness of B. Lowe's third definition is non-injective agreement, which implies three requirements.

1. A party A achieves weak agreement for B.
2. The run on B, in which the weak agreement is achieved, is of the role program that is expected by A. This property is in fact implied by the next requirement.
3. The value of a variable set ds in the role program of A is the same as the value of ds in the run on B.

The third requirement is not directly related to entity authentication. The first two requirements are roughly comparable to the case when A achieves both identification and willingness FLAGs for B. The canonicity requirement in the SI methodology is implied by the third requirement. A non-injective agreement on ds implies that the values assigned to ds are canonical messages.

Lowe's fourth definition is injective agreement, which additionally requires one-to-one relationship between the runs of A and B. Ignoring the message agreement part, this goal roughly corresponds to one-sided authentication (identification, willingness, and operativeness).

The notion of time is captured in Lowe's recentness goal. He describes two methods to achieve recentness: by an agreement on a fresh data (nonce) or by using timed authentication. The first method is comparable to our operativeness goal, since operativeness requires a mapping between a nonce (fresh data) and a binding sequence. The second method relies on an additional (global) process that counts time ticks. This method seems to be implementation dependent and it is not clear to us how to capture timed authentication from the local perspective of a party.

In comparison to Lowe's definitions, our definitions of FLAGs have two advantages. First, FLAGs are services oriented properties of entity authentication, and do not include message agreement requirements, which may not be relevant to entity authentication. Second, FLAGs are non-security requirements, which are validated using a single security property, viz canonicity. The four definitions by Lowe are all security requirements.

Synchronization

Cremers [53] introduces two more authentication goals: non-injective synchronization and injective synchronization. The notion of synchronization is closely

related to Lowe's agreement. Synchronization between two parties on a set of message variables require that the two parties agree on the values of message variables and that the two parties have sent and received those values in the same order as specified in the protocol. Cremers shows that synchronization is a stronger requirement than Lowe's agreement. He also provides several examples where violation of synchronization leads to some unexpected effects. Depending on the construction of a protocol, this goal may be crucial for achieving a certain FLAG, but correct ordering of messages is not a general authentication goal.

Other Definitional Work

Syverson et al. propose six goals of authentication protocols [151, 149]. Two of them are related to entity authentication while the rest are related to key establishment. The first authentication goal is ping authentication: a party A achieves ping authentication of B if A believes that B has sent a message. This goal is comparable to our notion of identification and Lowe's notion of aliveness. The second authentication goal is called "entity authentication", which is achieved if a message received from B is fresh. This goal is comparable to the case when identification and operativeness goals are achieved together. Syverson et al. specify these goals in a belief logic. Boyd and Mathuria [40] point out that these definitions are not precise.

The concept of entity recognition has been used in many existing works [139, 7, 87]. Lucks et al. [99] show, using a complexity theoretic proof, that the Jane-Doe protocol achieves entity recognition. Their operational definition of "A recognizing B" is in the form of recoverability properties: if A receives a message from a B then A accepts the message; if A receives a message that is not from B then A rejects the message.

Boyd and Mathuria [40] propose a hierarchy of authentication goals. The hierarchy includes only two goals related to entity authentication: once authenticated, which means a party A once has had knowledge of a party B as her peer entity; and far-end operative, which means B is currently there. Clearly, these goals correspond to the notions of identification and operativeness.
Focardi et al. [70] compare three definitions of authentication: their own non-interference based GNDC authentication, Lowe's agreement [97], and spi-calculus based authentication [2]. They show that agreement is a stronger notion than the other two.

Guttman and Thayer [84] introduce the notion of *authentication tests* using a strand space based security model. They formalize three types of authentication tests. First one is called an outgoing test, in which a message is sent in an

encrypted form such that only a certain party can decrypt it, e.g., A sends a message encrypted using the public key of B and later A receives the same message. The second one is called an incoming test, in which a message is received in an encrypted form and only a certain party can perform that encryption. The third one is unsolicited test, which combines the ideas of the first two tests. These tests provide non-injective agreement on a message. If the message is a nonce then an injective agreement is achieved.

8.2 Analysis of Authentication

In general, deriving security properties of protocols is an undecidable problem [110]. This means that one cannot hope to find an algorithm (which can be implemented in a computer) that produces proofs of authentication goals for an arbitrary authentication protocol. This undecidability result, however, does not prevent constructing manual proofs of security for a given protocol, as we do in the SI methodology. In formal security models, there are various ways to get around this problem. Some algorithms produce false positives [34], others miss some attacks or do not terminate [16]. Typically, there are many boundary conditions, such as maximum number of sessions. Some algorithms ask for human intervention when producing a proof [126].

In this section, we briefly look into to the existing methods and tools available for the analysis of authentication protocols. Most of these techniques can be used as part of the SI methodology to validate canonicity.

8.2.1 Authentication Logic

An important line of work for the verification of authentication properties is authentication logic. Burrows, Abadi, and Needham invented the so-called BAN logic [44]. This type of logic is also called belief logic, because the predicates are on the beliefs of protocol parties, such as a party A believes in a statement X. The predicates are derived using a set of logic rules, e.g., if A sees a message signed by a party B then A believes that the message was once sent by B.

A proof in the BAN logic is usually intuitive and simple. There are four main steps in an analysis that is based on BAN logic. First, an authentication protocol is transformed to logic formulas, which is called the idealisation step. Second, the assumptions about the initial beliefs of the protocol parties are specified. Third, each flow of the idealised protocol is annotated with the assertions about

the states of the protocol parties. In the last step, the rules of BAN logic are used to derive the beliefs held by the protocol parties. If the derived beliefs are consistent with the intended goals of the protocol then the protocol is considered secure.

This approach does not prescribe a definition of entity authentication, but one can formulate authentication goals in form of beliefs, in a protocol dependent manner. Syverson and Cervesato [149] provide comprehensive survey of expressing authentication goals in various belief logics. A few examples of FLAGs in BAN logic are as follows:

- A party A achieves the identification of a party B if A believes that B once said a message M.
- A party A achieves the identification and operativeness of a party B if A believes that B once said a message M, and A believes that M is fresh.
- A party A achieves the identification and willingness of a party B if A believes that B once said a message M, and A believes that B believes that A receives M.

Note that the above definitions of FLAGs are in terms of protocol messages. These can be compared to our definitions of FLAGs, which are abstract and are independent of the details of an authentication protocol. To the best of our understanding, BAN logic cannot be used to prove canonicity of messages. On the other hand, BAN logic supports many other security goals that are not covered by the structured intuition. For example, it is straight forward to specify key distribution goals in BAN logic, as a belief in a session key and the fact that the peer entity also believes in the session key.

Besides the definitional aspect, this line of work has a few methodological limitations [118], which are also evident from the "correctness proof" of Needham-Schroeder public key (NSPK) protocol [44]. The NSPK was described in Chapter 6 and it is prone to a man-in-middle attack [96]. The BAN logic assumes that a party can detect and ignore his own messages, and the adversary is not a legitimate network party; that is why the man-in-middle attack on the NSPK protocol remained undetected.

The idealisation process of BAN logic, which converts a protocol specification to the statements of BAN logic, is an error-prone process, as exemplified by Boyd and Mao [39]. To address such limitations, many extensions of the BAN logic have been proposed [4, 153, 151, 147].

8.2.2 Reduction

Bellare and Rogaway pioneered this line of work [28]. They prove the security of a two-role authentication protocol in a complexity theoretic model. They use matching conversations as the definition of authentication. The overall approach is the same as the one used for proving the security of cryptographic algorithms, namely a reduction style proof in which the security of an authentication protocol is reduced to a well understood assumption. In this case, the assumption is the existence of pseudo random functions.

Bellare and Rogaway are also the first to formalize an insider adversary in their model of communication. An insider adversary is a legitimate network user. This adversarial capability was lacking in some of the earlier works, such as in the Dolev-Yao model [61] and the BAN model [44].

This approach is used in many subsequent analyses of authentication protocols, e.g., for a server based three-role protocol [26], public key based protocol [31], and password based protocol [41]. For the SI methodology, this approach means constructing a reduction type proof for the canonicity of messages. The main advantage is that canonicity is much weaker security requirement than matching conversations, which could make concrete security proofs considerably simpler.

8.2.3 Interactive Theorem Proving

This approach is semi-automated and is introduced by Paulson [126] using Isabelle/HOL theorem prover, and by Dutertre and Schneider [63] using PVS theorem prover. Paulson uses inductive style arguments to prove security properties. The main component in his approach is an inductive definition and the corresponding induction rule. An inductive definition captures an infinite sequence of communication events, such as a trace of sent and received messages. One can prove that a certain property holds for all possible traces generated by an inductive definition.

This method has been used for the verification of many large protocols, e.g., Kerberos [21], TLS [127], SET [20], and certified e-mail [19]. The main advantage of this approach is that the proofs are machine checkable. The proofs, however, are carried out under strong assumptions, e.g., perfect encryption and strictly typed variables. Therefore, these proofs are very different from complexity theoretic proofs. In this regard, an interesting case is discovered by Ryan and Schneider [134], who present an attack on a protocol that has an inductive security proof. For authentication, Paulson uses inductive arguments

to prove message authentication [126]. This means that this approach can be used to prove canonicity of a message, since canonicity is implied by message authentication.

8.2.4 Simulation

The notion of simulation is extensively used in cryptography, especially in the context of zero-knowledge proofs. In a simulation based analysis, two types of models are constructed for a protocol. One model is called an ideal model and the other is called a real model. The aim of a security analyst is to show that whatever an adversary could gain in a real model, could also be gained in the ideal model. This approach is independent of the protocol goals. In fact, the goals of a protocol are embedded in the construction of the ideal model. The notion of simulation can be formalized in different flavours, e.g., observational equivalence [94] and reactive simulatability [128, 46].

For authentication protocols, simulation is first time used by Bellare et al. [23]. Shoup [141] and Canetti et al. [48, 46] also use simulation based approach in the analysis of authentication protocols.
The main idea in the work of Bellare et al. [23] is to use so called *authenticators*. An authenticator is a universal compiler that emulates a protocol that executes in an ideal model into a protocol that executes in the corresponding real model. The ideal model assumes authenticated channels while the real model does not employ this assumption. Their work aims at solving the secure key exchange problem in a modular fashion: given a secure protocol of key exchange that executes on authenticated channels, one uses an authenticator to transform this secure protocol into another secure protocol that executes on unauthentic channels. They use this approach to get a secure key exchange protocol using the Diffie-Hellman (DH) protocol; the DH protocol is secure on authentic channels.

Although Bellare et al. only capture message authentication requirement, their approach is quite promising for designing new protocols using the SI methodology. In its original form, one can completely analyse a proposed protocol assuming that canonical messages are communicated over authentic channels, without any regard to the network adversary. Then, the protocol can be transformed into a secure protocol using an authenticator. It may also be possible to formulate a weaker version of an authenticator that only provides a canonical channel.

8.2.5 Strand Space

Fábrega et al. [68] develop the strand space approach. A strand is similar to a process in a message sequence chart. A strand is a linear structure representing a sequence of sent and received messages. The sequence of messages in a role program constitutes a legitimate strand. There can be a number of adversarial strands. A collection of strands is called a bundle, which is a similar concept to our notion of a session, except a session only consists of runs of role programs while a bundle may have adversarial strands. A bundle is a graphical structure in which there are two types of arcs, one for strand succession and the other for message transmission.

Although there are many notational similarities, a bundle is very different from an instance of a dependency graph:

- A bundle represents dynamic behaviour of a protocol execution. An instance of a dependency graph represents functional relations between messages.
- A node of a bundle and the value of a dependency node are the same thing, but a bundle does not have an equivalent notion of interim nodes. More importantly, an arc (arrow) in a bundle represents a causal precedence between two messages, while an arc of a dependency graph represent a functional relation between two messages. We do not capture such causal precedence in our model, and strand space does not take functional dependencies into account.
- A bundle is used to prove security properties, while a dependency graph is only used in non-security analysis. Dependency analysis only helps in specifying the security (canonicity) requirements and is essential for our correctness analysis.

Lowe's agreement can be used as a definition of authentication in the strand space model. Cremers [54] points out that strand space is tightly coupled with Dolev-Yao attacker model and synchronization property cannot be immediately specified. There are many automated tools that are based on the strand space model, e.g. Athena [142], Constraint Solver [108], ASPECT-a [115].

Within the strand space model, Guttman, Doghmi, and Thayer develop a theory of skeletons and shapes [60], with an aim to capture the complete behaviour of a protocol. A skeleton is a graphical structure that captures three aspects of protocol specifications: sequence of message in a role program (strands), secret keys, and nonces. A shape of a skeleton is a realizable structure that is compatible to the skeleton. A shape represents a possible execution of the protocol in absence of any adversary. The idea is that a skeleton (the protocol

model) usually corresponds to a finite number of shapes. Therefore, by analysing the set of all shapes, one can guarantee secrecy and authentication properties. The authors provide a procedure that determines the shapes of a protocol and they develop a software called cryptographic protocol shape analyser.

8.2.6 Fully Automated Analysis

The first attempt for an automated analysis of authentication protocol was by Millen using Prolog [109]. Meadows [105] presents an extensive survey in this area. In 1996, Lowe developed a method to specify authentication protocols in CSP (communicating sequential processes) and then he uses a general purpose model checker FDR to verify its security properties. He discovered the attack against NSPK protocol (described in Chapter 6), which attracted a lot of attention towards the use of automated tools in security analysis. An automated tool for security analysis is based on various algorithms. Different algorithms have different features:

Verification/Falsification: Some algorithms attempt the verification of a security property, e.g., in NRL [104], ProVerif [32], and LySa [34]. Some algorithms attempt falsification of a security property, namely they try to find attacks, e.g., FDR [75] and OFMC [16].

Technique: Most of the tools use model checking. FDR [75] is a model checker for CSP process algebra. NRL [104] uses a backward search algorithm to find out whether insecure state can be reached from the initial state; NRL [104] is written in Prolog and is a hybrid tool containing a model checker and a theorem prover. ProVerif [32] represents protocol steps as Horn clauses and applies resolution-based theorem proving on them. LySa uses static program analysis to verify confidentiality and authenticity properties of protocol messages. The Constraint Solver [108], which is developed by Millen and Shmatikov, translates a security goal from strand space to a constraint solving problem. OFMC [16] uses the concept of a lazy intruder [14] and constraint differentiation [15] for an efficient forward model checking.

General/Special: FDR [75] and Murφ[111] are general purpose algorithms while OFMC [16], Scyther [53], and ProVerif [32] use special purpose algorithms.

Approximation: Many verification algorithms use over-approximation, namely they can generate false attacks [32] or may fail to prove secure protocols [34].

Availability: NRL [104] and Athena [142] are not freely available.

8.2.7 Adaptable Security Analysis

The term Quality of Protection (QoP) is also used to describe adaptable security models. Security and performance trade-offs in client-server environments are considered in Authenticast [137], which is a dynamic authentication protocol. The adaptation is due to flexible selection of key length, algorithm and the percentage of total packets that are authenticated. Ong, et al.[121], address the problems introduced by the traditional view of security—a system is either secure or insecure—by defining different security levels based on key size, block size, type of data and interval of security. Hager [85], in his thesis, considers the trade-offs of security protocols in wireless network; the security adaptation is on the basis of performance, energy, and resource consumption. Covington, et al.[52], propose parametrised authentication, in which quality of authentication is described in terms of sensor trustworthiness and the accuracy of the measurements. Lindskog [95] developed some solutions in his thesis for tuning security for networked applications. The proposed methods, however, are limited to confidentiality. Instead of using just one instance of authentication protocol, in some approaches, e.g., by Ganger [72], over a period of time a system can fuse observations about the entities into a kind of probabilistic authentication.

Many of the proposed RFID protocols [9, 114] are too heavy for low cost tags, and not supported by EPCGen2 [66]. Burmester, et al. [43] report that five different proposals that are compliant to EPCGen2 but have some security vulnerabilities. Damgård et al. [55] study the trade-offs between complexity and security using secret key cryptography. They propose a weaker but more practical notion of privacy; strong privacy requires a separate and independent key for each of the RFID tags. Vaudenay [154] uses eight different attacker models to reason about the privacy of RFID identification protocols. The strongest notion of privacy in Vaudenay's model is shown to be impossible to achieve; even the two other strong models mandate the use of public key cryptography. This model also serve as an inspiration for much of the following work where the authors use a class of attacker models, e.g., Paise et al.[122], Yu Ng et al. [120] and Canard et al.,[45].

Bella identifies the limitations in Dolev-Yao attacker model and propose the BUG threat model [18]. In BUG there are three type of entities: a good one who does not break the rules; a Dolev-Yao attacker; and an ugly one with no specific commitment. Classically, when an attack is found, the protocol is considered broken, but if we use a BUG style threat model then according to Bella one can still retaliate, which can save some re-engineering cost [17].

Ksieżopolski, et al. [92] describe the problem of an unnecessarily high level of security that can affect system dependability; they present an informal model of adaptable security, which, however, is difficult to justify for the soundness of results. Sun, et al. [148], propose an evaluation method for QoP, based on normalized weighted tree.

8.3 Other Related Work

The first effort to solve the problem of authentication in a network environment is in the seminal paper of Needham and Schroeder [117] in 1978. The design rationales used by Needham and Schroeder motivated the development of many other similar protocols, including Kerberos [119]. Needham and Schroeder assume an intruder that essentially controls the network and use abstract cryptographic functions. These notions were later formalized in 1983, in the well known Dolev-Yao model [61], which employs symbolic encryption and an adversarial control network. Needham and Schroeder, at that time, perceived the complex nature of seemingly simple protocols, and encouraged systematic analysis techniques to avoid subtle errors.

Needham and Schroeder (NS) protocols have some flaws, which were discovered after many years. Two of the attacks are most notable. The NS protocol that is based on symmetric key is vulnerable to a replay attack if any of the old session keys are compromised [57]. The second attack [96] is more interesting, mainly because it was published after 17 years and it first time uses an automated tool to discover a major attack. We have described this attack in Chapter 6.

These two attacks on the NS protocols are quite instructive for a protocol designer because these attacks use assumptions that were not stated in the original NS paper. The first attack [57] only works if an old session key is compromised. The second attack only works if an adversary is a legitimate network party. Further, the second attack also depends on the definition of authentication, as we described in Chapter 6. There are many other attacks on these protocols that exploit the unstated assumptions of the NS protocols [36, 5].

As pointed out by many authors [78], only a violation of the service requirements of entity authentication should be called a valid attack. For instance, the multiplicity error of DSSK protocol [98] is not a valid attack because it does not violate the claimed goals [57], namely neither confidentiality of the session key nor the entity authentication of participants is violated. Similarly, not all reported attacks have the same significance for security. For instance, a reported type flaw [50] is based on a somewhat dubious assumption:

if $\{T\} \equiv \{T, \{B, K_{AB}, T\}_{SA}\}$. Even if this assumption holds, the session key remains confidential and there is no violation of authentication.

Gong [80] proposes the use of one-way functions in construction of authentication protocols. She observes that authentication does not require that cryptographic functions ensure confidentiality of messages. Mao and Boyd [101] also advocate such use of one-way functions. These ideas may be considered as a pre-cursor to our notion of a dependency function.

Pancho [123] points out that, unlike protocol messages, protocol goals and assumptions are not often completely explicit. Park et al. [124] propose a classification scheme for authentication protocols, which could be helpful in defining different types of authentication protocols. Gollman [78] proposes the differentiation between packet switching and circuit switching networks when specifying an authentication protocol and emphasizes the difference between two unilateral-authentications and mutual-authentication.

The security analysis in the SI methodology can be carried out in a traditional cryptographic style, but a cryptographic analysis is done by hand, and the support of automation [13] is quite limited. Unlike cryptographic schemes, a cryptographic model of any reasonably sized protocol is often complex. Due to the painstaking work involved in concrete security analysis, only a handful of network protocols have been analysed, e.g., only a small fraction of roughly 200 protocols listed in 2003 [40] are accompanied by such security analysis.

The use of a formal security model helps in quickly validating the canonicity requirements of a protocol, but this comes at a price: any security assurance in a symbolic model does not automatically translate to the underlying computational cryptography and, therefore, to its hardware/software implementation. This is due to a huge gap between cryptographic assumptions and the assumptions behind the symbolic abstractions. An impressive amount of research has been done for establishing a theoretically sound link between symbolic cryptography and complexity-theoretic cryptography [3, 107]. In the line of universal composability, Ran Canetti and Herzog [47] show that the Dolev-Yao model can be layered on top of the traditional universal composability framework. Currently, this approach is limited to so-called simple protocols: the protocols that use only those cryptographic schemes that have some standard symbolic counterparts.

An adversary model is an essential part of a security analysis. There are two aspects of an adversarial model. The first one is related to the abstraction of the model. The most popular abstraction is the Dolev-Yao model [61]. In this symbolic model, two types of simplifications are introduced. First, binary strings and functions are replaced by symbolic terms and derivation rules. In

particular, this results in idealized encryption functions—either an adversary can decrypt a symbolic ciphertext (e.g., if he can derive the key) or the adversary gets absolutely no information about the plaintext. The second simplification is related to the computational capabilities of an adversary, namely the adversary is modelled as a non-deterministic strategy that is limited to selecting its actions from a small set of (pre-defined) logic rules. The security models that use these two abstractions are commonly referred to as symbolic/formal security models. Alternatively, one can use a cryptographic model in which an adversary is assumed to be able to compute any polynomial time function.

The second aspect is the network related capabilities of an adversary. In the Dolev-Yao attacker model [61], an adversary controls the communication network and is only constrained by cryptographic schemes, which are assumed to be ideal black boxes. Later on, a more powerful attacker model, where the adversary could be an insider, is used in the definitions by Bellare and Rogaway [28].

An important practical concern arises during the instantiation of encryption. The dependency relations of a protocol can change, e.g., if a security analyst assumes the use of a cipher (pseudo random permutation) for an encrypted message while a system developer uses the CBC mode of encryption [64]. Since the use of a cipher is assumed in a typical protocol narration, a safe option is to use authenticated encryption [24].

The requirements of a cipher are more stringent than an encryption scheme, because a cipher does not rely on a random initializing vector (IV) and the output of a cipher is non-malleable. In this regard, a variable length cipher [27] and an online cipher [22] are promising options. In an online cipher, encryption of a plaintext block only depends on the current block and the previous blocks of the plaintext. Without a random IV, CBC and CFB modes are the candidates of online cipher, for which Fouque et al. [71] show that the CFB mode is secure and the CBC mode is not secure. The CBC mode is only secure with a randomly chosen IV [76].

The problems that arise from the instantiation of encryption can affect any security analysis that relies on an abstract notion of encryption. One advantage of the SI methodology is that the dependency analysis, security analysis, and correctness analysis can be done using different levels of abstraction, including computational models. Moore [116] was probably the first to highlight the security problems that may occur in implementing symbolic encryption. Mao and Boyd [102] discuss some general vulnerabilities. Bellovin [29] reported vulnerabilities in the earlier versions of IPsec by exploiting CBC-mode encryption. An interesting case is that of encryption-only-mode of IPsec, for which Paterson and Yau [125] exploited CBC mode of encryption. Their attacks work if an implementation does not follow the standard strictly. Later, Degabriele

and Paterson [56] published another attack that works only if an implementation strictly follows the standard. These examples show that implementation choices are security critical for authentication protocols.

8.4 Summary

In this chapter, we examined the state of the art that is related to entity authentication protocols. The existing definitions of entity authentication are not satisfactory. Some definitions are informal without their formal interpretations. Many formal definitions are without their service-oriented interpretations. Some formal definitions only capture a generic feature of a protocol, which may or may not be relevant to entity authentication depending on the actual protocol. Some formal definitions are specific to a security model and a certain level of abstraction. Sometimes authentication goals are entangled with the goals that are not related to entity authentication. We described some of the existing techniques used for a protocol analysis. Most of these techniques can be used with the SI methodology to verify the canonicity of messages. An advantage of using the SI methodology is that the canonicity is a less stringent security requirement, which makes the existing techniques of protocol analysis more effective.

Chapter 9

Conclusions and Future Work

In this chapter, we draw conclusions, summarize our contributions reported in the previous chapters, and indicate a few directions for future work.

9.1 Conclusions

In this thesis, we presented the structured intuition (SI), which is a high level methodology that can be used to analyse a protocol for fine level authentication goals (FLAGs). To a system developer, entity authentication is a service, on which a larger system relies on to authenticate network parties. An SI based analysis is for the service oriented requirements of entity authentication, which are captured in the conceptual definitions of FLAGs. In the past, ambiguity in the meaning of authentication has led to many disagreements about the notion of security. For instance, an authentication protocol of ISO/IEC 9798-3 standard is insecure under one definition [28], due to the Canadian attack [40], but the protocol is secure under a different definition of authentication [97]. Therefore, the definitional effort behind the formulation of FLAGs is well justified.

The SI methodology is not tied to a particular level of abstraction, such as Dolev-Yao model [61] or Bellare-Rogaway model [28]. Also, different steps of the SI can be carried out with different level of abstractions. A methodological step of the

SI can be individually refined, which allows incremental refinements of analysis results. For instance, a D-graph can be constructed with an assumption of idealized symmetric encryption in an initial design phase, which can be refined to symbolic modes of encryption [5], which can be further refined to an encryption model that is based on pseudo random permutations. Similarly, canonicity of messages can be asserted using different methods, such as model checking or a reduction type cryptographic proof.

This flexibility in the level of abstraction allows one to analyse a protocol specification that is closer to actual implementation. For instance, in one of our works, we specify modes of encryption and then try to verify authentication goals of Denning-Sacco protocol [58]. Our analysis indicates some serious vulnerabilities, which we were able to exploit and are reported elsewhere [5].

In the SI methodology, a security analysis is demarcated from non-security analysis. In general, for a cryptographic protocol, correctness defines the purpose of the protocol, e.g., the output should follow a certain distribution (multi-party computation), some assurance on who participated in the protocol (authentication), and an assurance on who else may know the input or output of the protocol (secrecy). Security, on the other hand, is the assurance that the protocol remains correct in the presence of an adversary. Therefore, correctness and security are distinct concepts, but it is not always possible to keep this distinction in a security model. Only a few other security models [46] demarcate security as we do in the SI methodology. The job of a security analyst (human/automated tool) should be less strenuous if security requirements are fewer and pure, considering the security analysis is an undecidable problem in general [67, 110]. In the SI, the only property that need to verified against a network adversary is canonicity. In a sense, the SI methodology reduces the entity authentication problem to the verification of canonicity.

An authentication protocol is designed with certain trust and environment assumptions. In many cases, a protocol is used for many years to come [8]. As the system around an authentication protocol evolves with the introduction of new services over time, so does the security requirements and trust assumptions of the protocol. A traditional security model has some sort of binary output (for a system developer), namely whether a protocol is secure or insecure. If a protocol is marked as insecure then the protocol should not be used. An adaptable security analysis can provide information such as to what extent an "insecure" protocol is insecure, and which additional assumptions can make the protocol secure. This type of information can help in a cost-benefit analysis that determines whether a new protocol should be deployed or the "insecure" protocol is tolerable.

The SI methodology makes it easier to do an adaptable security analysis, where

adaptability is in terms of FLAGs, attacker model, and other environment assumptions. Adaptability is also useful in those applications where one needs a trade-off between security and resource requirements. Economics of security plays an important role in a competitive business environment, and not all real world applications require security against an all powerful attacker. A high level of security, even if it is necessary, could be unaffordable. It must be noted that an adaptable analysis does not change the actual security of a protocol; the analysis just captures the weak form of security which is otherwise labelled as insecurity in a tradition model. System designers, therefore, must be cautious while interpreting the results obtained in an adaptable analysis; security guarantees are accompanied by extra assumptions and a weaker adversary model, which may not be justifiable in a different application environment.

In summary, the advantages of using the SI methodology are as follows. The SI employs better definitions of entity authentication. The result of a correctness analysis, in terms of FLAGs, is fine grained. A complete analysis highlights protocol structure and design rationales. An SI based analysis is intuitive and therefore errors in the arguments are easier to locate. The analysis can be carried out with different level of abstractions. The methodology can be used with an application specific attacker model. It is easier to do an adaptable security analysis. The results for a typical authentication protocol can be obtained relatively fast.

9.2 Contributions

Our main contribution is a new methodology for the analysis of entity authentication protocols. The methodology is a workable solution, and it is based on various novel concepts, such as a dependency function, dependency graph, propagation of authenticity, canonical message, binding sequence, service-oriented FLAGs, and operational FLAGs. A summary of our contributions is listed in Table 9.1 at the end of this section.

We introduced the notion of a dependency function in Chapter 3, where we demonstrated that it is an important tool for the analysis of entity authentication protocols. To the best of our knowledge, no other verification methodology makes use of this abstraction in the manner we do. We also introduced the concept of a global D-graph, which provides a static model of a protocol. A global D-graph is useful in its own right, because it represents a legitimate session of the protocol. Further, authenticity propagates on a D-graph. The structure of a global D-graph may help a protocol designer to specify implementation details, locate critical functions, and remove redundant messages.

We introduced an execution environment of a protocol in terms of its role programs (Ch. 4). We modelled a role program as a set of local D-graphs. We introduced a new security property, viz. canonicity (Ch. 4), which is the only requirement that needs to be validated in the presence of a network adversary. This means that canonicity is a security property and depends on the dynamic behaviour of a protocol. We introduced the notion of a binding sequence (Ch. 4) that is constituted by canonical messages that are interconnected by a local D-graph. This is a new notion, which can directly be used to derive authentication properties. For a binding sequence, a local D-graph plays three important roles. First it is used to find out which of the protocol messages need to be canonical. Second, it justifies the propagation of canonicity to other protocol messages. Third, it links together different canonical messages so that it is guaranteed that they are all assigned in a single partial session.

We provided a comprehensive definition of entity authentication in terms of FLAGs (Ch. 5). Two types of definitions were provided for each FLAG. The first type is a conceptual definition that captures the service-oriented requirement of a FLAG. Conceptual FLAGs constitute a hierarchy of authentication properties. The second type is an operational definition, for which we introduced a certain type of distinguishers. These distinguishers and D-functions provide a new way of formally interpreting service-oriented authentication goals. An operational FLAG specifies a procedures that is used to validate a service-oriented authentication goal from a binding sequence. As part of a case study, we derived the FLAGs that are achieved by a mutual authentication protocol.

We demonstrated that the definition of entity authentication plays an important role in determining protocol security, and that the SI methodology provides a detailed picture of actual security provided by a protocol (Ch. 6). For this purpose, we analysed two protocols that are insecure in a traditional sense. We used the SI methodology to analyse these protocols and showed that these protocols are not completely insecure.

We demonstrated how the SI methodology can be used in an adaptable manner (Ch. 7). We modelled an RFID based system and analysed a family of eight RFID based authentication protocols. Using our SI methodology, we found the necessary assumptions under which these protocols are secure. This demonstrates how a weak level of security, which is otherwise labelled as insecurity, can be captured in the SI methodology.

This table summarises the individual contributions to the structured intuition.

Dependency function	A new abstraction that models the functional relation between two messages of a protocol
Dependency graph	A novel graphical model that captures functional relations of a protocol and a role program; Highlights the design rationales of a protocol; Provides a unique insight into the structure of a protocol; An operational interpretation of a protocol session
Canonicity	A new security property, which is important for message-passing authentication protocols; A weaker requirement than the message authenticity; The only security property required for entity authentication goals
Binding Sequence	A new concept for the messages of a role program, which combines the properties of a D-graph and a canonical message; Provides a bridge between security and entity authentication; Can be used both as a goal (for the security analysis) and as an assumption (for the correctness analysis)
Conceptual FLAGs	A new service-oriented hierarchy of entity authentication goals (Although individual FLAGs appear in the literature, sometimes informally, sometimes implicitly, and sometimes entangled in a larger goal. Our contribution relates to identifying them and collecting them in a single hierarchy.)
Operational FLAGs	Novel operational interpretations of service-oriented authentication goals; Only depend on the security property of a binding sequence; Require a non-security analysis for their validation, which is usually an easier task than a security analysis
Methodology	Synthesizes all of the above ideas into a single workable solution; Enables the validation of FLAGs; Does not depend on a particular attacker model or a particular level of abstraction; Demarcate security and correctness requirements; Formulated as a step by step procedure, which makes it easier to do an adaptable analysis of a protocol

Table 9.1: Summary of Contributions

9.3 Future Work

There are many different directions in which the future work can be carried out. A detailed study of different steps of the SI methodology is required to place the methodology on a sound mathematical foundation. For this purpose, one could formulate a dependency graph in more concrete terms that explicitly mentions the probability of errors. It would be interesting to explore how the probability of errors is propagated on a D-graph and how much it affects the advantage of a computationally bounded adversary. One could also develop techniques that help in obtaining a complexity theoretic proof of canonicity.

The list of FLAGs that is presented in this thesis corresponds to commonly expected authentication goals. Another possible direction is to formulate other related goals, especially goals related to (session) key establishment and privacy. The confidentiality of a session key is a primary goal, but usually there are additional requirement, such as freshness of a session key and the assurance that protocol parties know that a session key has been established. It seems that a session key must be a part of a binding sequence. It may be possible to recast various FLAGs to a key, such as operativeness as fresh key, willingness as the consent of a far-end party to use a particular key. Similarly, one could formulate additional correctness goals related to privacy of protocol parties, such as untraceability and unlinkability.

In this thesis, we have only shown examples of two-role and three-role protocols. Another research direction is to validate the methodology with group authentication protocols, where the number of parties are large and may be not known in the start of a session. We have formulated our methodology with an arbitrary m but validation with m-role protocols, for $m > 3$, may expose some constraints. Perhaps, a more efficient formulation of the methodology could be developed.

Another research direction is towards a tool support. The process of constructing a dependency graph is manual but it could be made automatic, at least at a symbolic level. One could design a graph-theoretic tool that generates a dependency graph from a protocol narration and finds the security requirements of the protocol. Such a tool could be be integrated with a model checker to carry out security analysis. In this way, the process up to the computation of a binding sequence could be automated. It seems that automating the subsequent process, namely the correctness analysis, is a bit harder; perhaps templates of distinguisher can be used with a level of human guidance.

For a system design, we envisage a catalogue that lists protocol assumptions and resource requirements (memory footprint, computational requirement, communication bandwidth, etc.), against the set of FLAGs that the protocol achieves.

A system designer can select a protocol that meets his requirements, without worrying about why a particular FLAG holds for the selected protocol. The validity of a FLAG can be guaranteed by a security analyst, who using the operational definition shows that the FLAG is achieved.

To conclude, we list some issues that are much harder to address. As typical with any hand written proof, the application of structured intuition is prone to human errors. The success of verification depends on the expertise of a security analyst, especially for complex protocols. Therefore, a subsequent review of the analysis is recommended to avoid errors and incompleteness.

We also point out that no verification methodology can solve pleasantness problem associated with authentication protocols, namely whether a system developer correctly identifies the authentication requirements of a larger system. The SI methodology provides necessary support for specifying fine level requirements assuming that the requirements are correct. One must not underestimate human errors nevertheless. For security-critical systems, sound operating procedures, a level of resilience, and multi layer security, are equally important.

The scope of entity authentication spans beyond cryptography. As Jøsang says, unless an identity itself is meaningful there can be no authentication [89]. The meaning of an identity, trust on the relation between an identity and an actual entity, trust to communicate with an authenticated entity, and similar issues are beyond the scope of our work. System security requires a holistic view of many different areas, such as cryptography, economics of security, implementation details, human behaviour, and trust management.

Appendix A

Verification of Canonicity

We use a model checker OFMC (ver. 2011c) to verify the canonicity of the four messages M_4, M_6, and M_7, and M_8 of our five-pass protocol, described in § 4.5. OFMC was originally written by Mödersheim, Drielsma, and Köpf, at ETH Zurich. The tool OFMC, along with the tutorial and examples, is available online:
`http://www2.imm.dtu.dk/~samo`
The specification of our five-pass protocol for OFMC is as follows:

```
Types: Agent A,B,s;
       Number RA,RB,RB0;
       Symmetric_key KAB;
       Function sk

Knowledge: A: A,B,s,sk(A,s);
     B: B,A,s,sk(B,s),pre;
      s: A,B,s,sk(A,s),sk(B,s),KAB

Actions:

A->B: A,B,RA
B->s: B,s,RB0,RA,A
```

```
s->B: s,B,{|RB0,KAB,A|}sk(B,s),{|RA,KAB,B|}sk(A,s)
B->A: B,A,{|RA,KAB,B|}sk(A,s),{|RB,RA|}KAB
A->B: A,B,{|RA,RB|}KAB

Goals:
A weakly authenticates B on {|RB,RA|}KAB
A weakly authenticates s on {|RA,KAB,B|}sk(A,s)
B weakly authenticates s on {|RB0,KAB,A|}sk(B,s)
B weakly authenticates A on {|RA,RB|}KAB
```

This specification consists of four sections. The section Types is used to declare different types of protocol terms. The field Agent contains the name of parties who execute the protocol. In OFMC, there is no special notation to mark a role program, i.e., at some places A means identity of A and at other places A means the role program A^ρ executed by A. It is not difficult to distinguish these two cases however. The function $sk(.,.)$ is used in the next section to define long term keys.

The next section Knowledge describes the constants (as per the SI notations) that are supplied to each role program. The role program A^ρ (which is denoted by A in the above specification) is supplied with the values of identities of protocol parties and the long term key $sk(A, s)$ shared between A and s. Similarly, the role program B^ρ is supplied with a list of constants. The server program S^ρ, which is denoted by s in the above specification, is supplied with the long term keys of A and B. The lower case letter, s, indicates that s is a trusted program and therefore an adversary cannot execute this program. The other two programs denoted by A and B can be executed by an adversary.

Next section Actions specifies the protocol narration. The last section specifies the goals that OFMC verifies. We specify four goals corresponding to the canonicity requirements of M_4, M_6, M_7, and M_8. The goal specification *weakly authenticates* stands for Lowe's non-injective agreement. A non-injective agreement on a message implies that the message is a canonical message.

OFMC (ver. 2011) consists of two back-end verification engines, which are called classic module and fixed-point module. The classic module performs a bounded verification, namely it does model checking on a fixed number of parallel sessions. The fixed-point module performs the verification for an unbounded number of sessions, using abstract interpretation and over-approximation. On the flip side, the fixed-point module works in a strictly typed model, and the classic module

may find typing attacks. We execute OFMC (ver. 2011c) on a file that contains the above specification. The classic module reports that the four protocol goals are achieved, within a bound of eight sessions. The fixed-point module reports that the four protocol goals are achieved for unbounded number of sessions. Thus, we conclude that M_4, M_6, M_7, and M_8 are canonical messages.

Appendix B

Abstracts of Published Papers

1. A Mechanism for Identity Delegation at Authentication Level

Authentication and access control are normally considered as separate security concepts that have separate goals and are supported by separate security mechanisms. In most operating systems, however, access control is exclusively based on the identity of the requesting principal, e.g., an access control mechanism based on access control lists simply verifies that the authenticated identity of the requesting principal is on the list of authorized users.

In this paper we propose a human-to-human delegation mechanism for nomadic users, which exploits the amalgamation of authentication and access control in most operating systems, by delegating privileges at the identity level. The complexity of classic delegation models, especially if they strictly follow the principle of least privileges, often leads to a poor usability, which motivates a user to circumvent the default delegation mechanism. On the other hand, the identity delegation makes good use of trust relationships among users of a particular environment and offers the possibility of improved usability. Although identity delegation might violate the principle of least privileges, in practice it could increase the over all security of a nomadic environment where users need

to delegate their duties frequently. The proposed mechanism is independent of the access control and the delegation event is only logged at the authentication level. Due to its improved usability, the motivation to share authentication tokens is reduced.

Naveed Ahmed, Christian D. Jensen
NordSec 2009

2. Definition of entity authentication

Authentication is considered a pre-requisite for communication security, but the definition of authentication is generally not agreed upon. Many attacks on authentication protocols are the result of misunderstanding of the goals of authentication. This state of affairs indicate limitations in theoretical understanding of the meanings of authentication. We provide a new insight in this direction and formalize it in CFPS (Common Framework for authentication Protocols' Specifications). CFPS provides a precise scope of definition for authentication protocols, which could make the design and analysis process more systematic.

Naveed Ahmed, Christian D. Jensen
IWSCN 2010

3. Adaptable Authentication Model

Most methods for protocol analysis classify protocols as "broken" if they are vulnerable to attacks from a strong attacker, e.g., assuming the Dolev-Yao attacker model. In many cases, however, exploitation of existing vulnerabilities may not be practical and, moreover, not all applications may suffer because of the identified vulnerabilities. Therefore, we may need to analyze a protocol for weaker notions of security. In this paper, we present a security model that supports such weaker notions. In this model, the overall goals of an authentication protocol are broken into a finer granularity; for each fine level authentication goal, we determine the "least strongest-attacker" for which the authentication goal can be satisfied. We demonstrate that this model can be used to reason about the security of supposedly insecure protocols. Such adaptability is particularly useful in those applications where one may need to trade-off security relaxations against resource requirements.

Naveed Ahmed, Christian D. Jensen
ESSoS 2011

4. Security of Dependable Systems

Security and dependability are crucial for designing trustworthy systems. The approach "security as an add-on" is not satisfactory, yet the integration of security in the development process is still an open problem. Especially, a common framework for specifying dependability and security is very much needed. There are many pressing challenges however; here, we address some of them. Firstly, security for dependable systems is a broad concept and traditional view of security, e.g., in terms of confidentiality, integrity and availability, does not suffice. Secondly, a clear definition of security in the dependability context is not agreed upon. Thirdly, security attacks cannot be modeled as a stochastic process, because the adversary's strategy is often carefully planned. In this chapter, we explore these challenges and provide some directions toward their solutions.

Naveed Ahmed, Christian D. Jensen
Book: Dependability and Computer Engineering
2011

5. Demarcation of Security in Authentication Protocols

Security analysis of communication protocols is a slippery business, many "secure" protocols later turn out to be insecure. Among many, two complains are more frequent: inadequate definition of security and unstated assumptions in the security model. In our experience, one principal cause for such state of affairs is an apparent overlap of security and correctness, which may lead to many sloppy security definitions and security models. Although there is no inherent need to separate security and correctness requirements, practically, such separation is significant. It makes security analysis easier and enables us to define security goals with a fine granularity. We present one such separation, by introducing the notion of binding sequence as a security primitive. A binding sequence, roughly speaking, is the only required security property of an authentication protocol. All other authentication goals, the correctness requirements, can be derived from the binding sequence.

Naveed Ahmed, Christian D. Jensen
SysSec 2011

6. Post-Session Authentication

Entity authentication provides confidence in the claimed identity of a peer entity, but the manner in which this goal is achieved results in different types of authentication. An important factor in this regard is the order between authentication and the execution of the associated session. In this paper, we consider the case of post-session authentication, where parties authenticate each other at the end of their interactive session. This use of authentication is different from session-less authentication (e.g., in RFID) and pre-session authentication (e.g., for access control.)

Post-session authentication, although a new term, is not a new concept; it is the basis of at least a few practical schemes. We, for the first time, systematically study it and present the underlying authentication model. Further, we show that an important class of problems is solvable using post-session authentication as the only setup assumption. We hope post-session authentication can be used to devise new strategies for building trust among strangers.

Naveed Ahmed, Christian D. Jensen
IFIPTM 2012

7. Towards Private-Key Symbolic Encryption

Symbolic encryption, in the style of Dolev-Yao models, is ubiquitous in formal security models. In its common use, encryption on a whole message is specified as a single monolithic block. From a cryptographic perspective, however, this may require a resource-intensive cryptographic algorithm, namely an authenticated encryption scheme that is secure under chosen ciphertext attack. Therefore, many reasonable encryption schemes, such as AES in the CBC or CFB mode, are not among the implementation options.

In this paper, we report new attacks on CBC and CFB based implementations of the well-known Needham-Schroeder and Denning-Sacco protocols. To avoid such problems, we advocate the use of refined notions of symbolic encryption that have natural correspondence to standard cryptographic encryption schemes.

Naveed Ahmed, Christian D. Jensen, Erik Zenner
ESORICS 2012

Bibliography

[1] H. Abadi and R. Needham. Prudent engineering practice for cryptographic protocols. In *IEEE Computer Society Symposium on Research in Security and Privacy*, pages 122–136. IEEE, 1994.

[2] M. Abadi and A. Gordon. Reasoning about cryptographic protocols in the spi calculus. *8th International Conference on Concurrency Theory (CONCUR'97), Warsaw, Poland*, pages 59–73, 1997.

[3] M. Abadi and P. Rogaway. Reconciling two views of cryptography. *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics, Lecture Notes in Computer Science: Volume 1872/2000,*, pages 3–22, 2000.

[4] M. Abadi and M.R. Tuttle. A semantics for a logic of authentication. In *Proceedings of the tenth annual ACM symposium on Principles of distributed computing*, pages 201–216. ACM, 1991.

[5] N. Ahmed, C. Jensen, and E. Zenner. Towards symbolic encryption schemes. *European Symposium on Research in Computer Security, (ESORICS–2012), Lecture Notes in Computer Science, Volume 7459/2012*, pages 557–572, 2012.

[6] R. Anderson and R. Needham. Programming Satan's computer. *Computer Science Today, Lecture Notes in Computer Science: Volume 1000/1995*, pages 426–440, 1995.

[7] J. Arkko and P. Nikander. Weak authentication: How to authenticate unknown principals without trusted parties. In *Security Protocols, Lecture*

Notes in Computer Science: Volume 2845/2004, pages 57–66. Springer, 2004.

[8] N. Asokan, V. Niemi, and K. Nyberg. Man-in-the-middle in tunnelled authentication protocols. In *Security Protocols, Lecture Notes in Computer Science: Volume 3364/2005*, pages 28–41. Springer, 2005.

[9] G. Avoine and P. Oechslin. A scalable and provably secure hash-based RFID protocol. In *Third IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom–2005)*, pages 110–114. IEEE, 2005.

[10] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 220–230. ACM, 2003.

[11] L.C. Baird, W.L. Bahn, M.D. Collins, M.C. Carlisle, and S.C. Butler. Keyless jam resistance. In *IEEE Information Assurance and Security Workshop (IAW'07)*, pages 143–150. IEEE, 2007.

[12] B. Barak, R. Canetti, Y. Lindell, R. Pass, and T. Rabin. Secure computation without authentication. In *Advances in Cryptology–CRYPTO 2005, Lecture Notes in Computer Science: Volume 3621/2005*, pages 361–377. Springer, 2005.

[13] G. Barthe, B. Grégoire, S. Heraud, and S. Béguelin. Computer-aided security proofs for the working cryptographer. *Advances in Cryptology–CRYPTO 2011, Lecture Notes in Computer Science: Volume 6841/2011*, pages 71–90, 2011.

[14] D. Basin. Lazy infinite-state analysis of security protocols. *Secure Networking—CQRE [Secure]'99, Lecture Notes in Computer Science: Volume 1740/1999*, pages 781–781, 1999.

[15] D. Basin, S. Mödersheim, and L. Vigano. Cdiff: a new reduction technique for constraint-based analysis of security protocols. In *Proceedings of the 10th ACM conference on Computer and communications security*, pages 335–344. ACM, 2003.

[16] D. Basin, S. Mödersheim, and L. Vigano. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[17] G. Bella. What is correctness of security protocols? *Journal of Universal Computer Science*, 14(12):2083–2106, 2008.

[18] G. Bella, S. Bistarelli, and F. Massacci. Retaliation: Can we live with flaws? *NATO Security Through Science Series D-Information and Communication Security*, 6, 2006.

[19] G. Bella, C. Longo, and L. Paulson. Verifying second-level security protocols. *Theorem Proving in Higher Order Logics, Lecture Notes in Computer Science: Volume 2758/2003*, pages 352–366, 2003.

[20] G. Bella, F. Massacci, and L.C. Paulson. An overview of the verification of set. *International Journal of Information Security*, 4(1):17–28, 2005.

[21] G. Bella and L. Paulson. Kerberos version iv: Inductive analysis of the secrecy goals. *Computer Security—ESORICS 98, Lecture Notes in Computer Science, Volume 1485/1998*, pages 361–375, 1998.

[22] M. Bellare, A. Boldyreva, L. Knudsen, and C. Namprempre. Online ciphers and the hash-CBC construction. In *Advances in Cryptology—CRYPTO 2001*, pages 292–309. Springer, 2001.

[23] M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 419–428. ACM, 1998.

[24] M. Bellare and C. Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Advances in Cryptology—ASIACRYPT 2000*, pages 531–545, 2000.

[25] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[26] M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 57–66. ACM, 1995.

[27] M. Bellare and P. Rogaway. On the construction of variable-input-length ciphers. In *Fast Software Encryption*, pages 231–244. Springer, 1999.

[28] Mihir Bellare and Phillip Rogaway. Entity authentication and key distribution. In *CRYPTO'93: Proceedings of the 13th annual international cryptology conference on Advances in cryptology*, pages 232–249, New York, NY, USA, 1994. Springer-Verlag New York, Inc.

[29] S.M. Bellovin. Problem areas for the ip security protocols. In *Proceedings of the Sixth Usenix Unix Security Symposium*, pages 1–16, 1996.

[30] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of two-party authentication protocols. In *Advances in Cryptology—CRYPTO'91*, pages 44–61. Springer, 1992.

[31] S. Blake-Wilson and A. Menezes. Entity authentication and authenticated key transport protocols employing asymmetric techniques. In *Security Protocols*, pages 137–158. Springer, 1998.

[32] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Computer Security Foundations Workshop, 2001. Proceedings. 14th IEEE*, pages 82–96. IEEE, 2001.

[33] B. Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, 2009.

[34] Chiara Bodei, Mikael Buchholtz, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Static validation of security protocols. *Journal of Computer Security*, pages 347–390, 2005.

[35] A. Bogdanov, L. Knudsen, G. Leander, C. Paar, A. Poschmann, M. Robshaw, Y. Seurin, and C. Vikkelsoe. Present: An ultra-lightweight block cipher. *Cryptographic Hardware and Embedded Systems-CHES 2007*, pages 450–466, 2007.

[36] C. Boyd. Hidden assumptions in cryptographic protocols. In *IEE Proceedings: Computers and Digital Techniques*, volume 137, pages 433–436. IET, 1990.

[37] C. Boyd. On key agreement and conference key agreement. In *Information Security and Privacy*, pages 294–302. Springer, 1997.

[38] C. Boyd. Towards extensional goals in authentication protocols. In *Proceedings of the 1997 DIMACS Workshop on Design and Formal Verification of Security Protocols*, 1997.

[39] C. Boyd and W. Mao. On a limitation of ban logic. In *Advances in Cryptology—EUROCRYPT'93*, pages 240–247. Springer, 1994.

[40] C. Boyd and A. Mathuria. *Protocols for Authentication and Key Establishmen (1st ed.)*. Springer, 2003.

[41] V. Boyko, P. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *Advances in Cryptology—Eurocrypt 2000*, pages 156–171. Springer, 2000.

[42] B. Briscoe, A. Odlyzko, and B. Tilly. Metcalfe's law is wrong-communications networks increase in value as they add members-but by how much? *Spectrum, IEEE*, 43(7):34–39, 2006.

[43] M. Burmester and J. Munilla. A Flyweight RFID authentication protocol. In *Workshop on RFID Security, RFIDSec*, 2009.

[44] Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. *ACM Transaction Computer Systems*, 8(1):18–36, 1990.

[45] S. Canard, I. Coisel, J. Etrog, and M. Girault. Privacy-preserving RFID systems: Model and constructions. *Eprint: IACR*, 21:22, 2010.

[46] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 136–145. IEEE, 2002.

[47] R. Canetti and J. Herzog. Universally composable symbolic security analysis. *Journal of cryptology*, 24(1):83–147, 2011.

[48] R. Canetti and H. Krawczyk. Analysis of key-exchange protocols and their use for building secure channels. *Advances in Cryptology—EUROCRYPT 2001*, pages 453–474, 2001.

[49] Y. Chevalier and M. Rusinowitch. Compiling and securing cryptographic protocols. *Information Processing Letters*, 110(3):116–122, 2010.

[50] Y. Chevalier and L. Vigneron. Automated unbounded verification of security protocols. In *Computer Aided Verification*, pages 125–171. Springer, 2002.

[51] J. Clark and J. Jacob. A survey of authentication protocol literature: Version 1.0, 1997.

[52] M. Covington, M. Ahamad, I. Essa, and H. Venkateswaran. Parameterized authentication. *Computer Security–ESORICS 2004*, pages 276–292, 2004.

[53] C. Cremers. The scyther tool: Verification, falsification, and analysis of security protocols. In *Computer Aided Verification*, pages 414–418. Springer, 2008.

[54] CJF Cremers. Scyther. *Semantics and Verification of Security Protocols, Thesis, University Press Eindhoven*, 2006.

[55] I. Damgård and M. Pedersen. RFID security: Tradeoffs between security and efficiency. *Topics in Cryptology–CT-RSA 2008*, pages 318–332, 2008.

[56] J.P. Degabriele and K.G. Paterson. Attacking the ipsec standards in encryption-only configurations. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 335–349. Ieee, 2007.

[57] D.E. Denning and G.M. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[58] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Commun. ACM*, 24:533–536, August 1981.

[59] Whitfield Diffie, Paul C. Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2:107–125, 1992.

[60] S.F. Doghmi, J.D. Guttman, and F.J. Thayer. Skeletons, homomorphisms, and shapes: Characterizing protocol executions. *Electronic Notes in Theoretical Computer Science*, 173:85–102, 2007.

[61] D. Dolev and A. Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.

[62] Danny Dolev and Andrew C. Yao. On the security of public key protocols. In *SFCS'81: Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*, pages 350–357, Washington, DC, USA, 1981. IEEE Computer Society.

[63] B. Dutertre and S. Schneider. Using a pvs embedding of csp to verify authentication protocols. *Theorem Proving in Higher Order Logics*, pages 121–136, 1997.

[64] M. Dworkin. Recommendation for block cipher modes of operation. methods and techniques. Technical report, DTIC Document, 2001.

[65] E. Emerson. The beginning of model checking: A personal perspective. *25 Years of Model Checking*, pages 27–45, 2008.

[66] EPC EPCglobal. Tag data standards version 1.3. *EPCglobal Standard Specification*, 2005.

[67] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. In *24th Annual Symposium on Foundations of Computer Science*, pages 34–39. IEEE, 1983.

[68] F.J.T. Fábrega, J.C. Herzog, and J.D. Guttman. Strand spaces: Why is a security protocol correct? In *Security and Privacy, 1998. Proceedings. 1998 IEEE Symposium on*, pages 160–171. IEEE, 1998.

[69] R. Focardi, R. Gorrieri, and F. Martinelli. Non interference for the analysis of cryptographic protocols. *Automata, Languages and Programming*, pages 354–372, 2000.

[70] R. Focardi, R. Gorrieri, and F. Martinelli. A comparison of three authentication properties. *Theoretical Computer Science*, 291(3):285–327, 2003.

[71] P.A. Fouque, G. Martinet, and G. Poupard. Practical symmetric on-line encryption. In *Fast Software Encryption*, pages 362–375. Springer, 2003.

[72] G.R. Ganger. Authentication confidences. In *Proceedings of 8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, page 169, 2001.

[73] G. Gilder. Metcalf's law and legacy. *Forbes ASAP*, 27, 1993.

[74] O. Goldreich. *Foundations of cryptography: Basic Applications*, volume 2. Cambridge Univ Pr, 2004.

[75] M. Goldsmith et al. Fdr: User manual and tutorial, version 2.77. *Formal Systems (Europe) Ltd*, 2001.

[76] S. Goldwasser and M. Bellare. Lecture notes on cryptography. *Summer course "Cryptography and computer security" at MIT*, 1999:1999, 1996.

[77] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of computer and system sciences*, 28(2):270–299, 1984.

[78] D. Gollmann. Authentication–myths and misconceptions. *Progress in Computer Science and Applied Logic*, 20:203–225, 2001.

[79] D. Gollmann. What do we mean by entity authentication? In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*, pages 46–54. IEEE, 2002.

[80] L. Gong. Using one-way functions for authentication. *ACM SIGCOMM Computer Communication Review*, 19(5):8–11, 1989.

[81] L. Gong. Variations on the themes of message freshness and replay-or the difficulty in devising formal methods to analyze cryptographic protocols. In *Computer Security Foundations Workshop VI, 1993. Proceedings*, pages 131–136. IEEE, 1993.

[82] A.D. Gordon and A. Jeffrey. Authenticity by typing for security protocols. *Journal of computer security*, 11(4):451–520, 2003.

[83] J.D. Guttman and F.J. Thayer. Authentication tests. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 96–109. IEEE, 2000.

[84] J.D. Guttman and F.J. Thayer. Authentication tests and the structure of bundles. *Theoretical Computer Science*, 283(2):333–380, 2002.

[85] C.T.R. Hager. *Context aware and adaptive security for wireless networks.* PhD thesis, Virginia Polytechnic Institute and State University, 2004.

[86] N. Haller. The s/key one-time password system. 1995.

[87] J. Hammell, A. Weimerskirch, J. Girao, and D. Westhoff. Recognition in a low-power environment. In *Distributed Computing Systems Workshops, 2005. 25th IEEE International Conference on*, pages 933–938. IEEE, 2005.

[88] A. Hiltgen, T. Kramp, and T. Weigold. Secure internet banking authentication. *Security & Privacy, IEEE*, 4(2):21–29, 2006.

[89] A. Jøsang, M.A. Patton, and A. Ho. Authentication for humans. In *Proceedings of the 9th International Conference on Telecommunication Systems (ICTS2001), Cox School of Business, Southern Methodist University*, 2001.

[90] A. Juels. RFID security and privacy: A research survey. *Selected Areas in Communications, IEEE Journal on*, 24(2):381–394, 2006.

[91] J. Katz and M. Yung. Complete characterization of security notions for probabilistic private-key encryption. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 245–254. ACM, 2000.

[92] B. Księżopolski and Z. Kotulski. Adaptable security mechanism for dynamic environments. *Computers & Security*, 26(3):246–255, 2007.

[93] L. Lamport. Password authentication with insecure communication. *Communications of the ACM*, 24(11):770–772, 1981.

[94] P. Lincoln, J. Mitchell, M. Mitchell, and A. Scedrovy. Probabilistic polynomial-time equivalence and security analysis. *FM'99–Formal Methods*, pages 708–708, 1999.

[95] S. Lindskog. *Modeling and tuning security from a quality of service perspective*. Chalmers University of Technology, 2005.

[96] G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. *Tools and Algorithms for the Construction and Analysis of Systems*, pages 147–166, 1996.

[97] G. Lowe. A hierarchy of authentication specifications. In *csfw*, page 31. Published by the IEEE Computer Society, 1997.

[98] G. Lowe et al. A family of attacks upon authentication protocols. 1997.

[99] S. Lucks, E. Zenner, A. Weimerskirch, and D. Westhoff. Concrete security for entity recognition: The Jane Doe protocol. *Progress in Cryptology-INDOCRYPT 2008*, pages 158–171, 2008.

[100] W. Mao. An augmentation of ban-like logics. In *Computer Security Foundations Workshop, 1995. Proceedings., Eighth IEEE*, pages 44–56. IEEE, 1995.

[101] W. Mao and C. Boyd. Development of authentication protocols: Some misconceptions and a new approach. In *Computer Security Foundations Workshop VII, 1994. CSFW 7. Proceedings*, pages 178–186. IEEE, 1994.

[102] W. Mao and C. Boyd. On the use of encryption in cryptographic protocols. *Codes and Cyphers: Cryptography and Coding IV*, pages 251–262, 1995.

[103] U.M. Maurer and P.E. Schmid. A calculus for security boots trapping in distributed systems. *Journal of Computer Security*, 4(1):55–80, 1996.

[104] C. Meadows. The nrl protocol analyzer: An overview. *The Journal of Logic Programming*, 26(2):113–131, 1996.

[105] C. Meadows. Formal methods for cryptographic protocol analysis: Emerging issues and trends. *Selected Areas in Communications, IEEE Journal on*, 21(1):44–54, 2003.

[106] A.J. Menezes, P.C. Van Oorschot, and S.A. Vanstone. *Handbook of applied cryptography*. CRC, 1997.

[107] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. *Theory of Cryptography*, pages 133–151, 2004.

[108] J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 166–175. ACM, 2001.

[109] J.K. Millen, S.C. Clark, and S.B. Freedman. The interrogator: Protocol secuity analysis. *Software Engineering, IEEE Transactions on*, (2):274–288, 1987.

[110] J. Mitchell, A. Scedrov, N. Durgin, and P. Lincoln. Undecidability of bounded security protocols. In *Workshop on Formal Methods and Security Protocols*, 1999.

[111] J.C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murφ. In *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on*, pages 141–151. IEEE, 1997.

[112] S. Mödersheim and L. Vigano. The open-source fixed-point model checker for symbolic analysis of security protocols. *Foundations of Security Analysis and Design V*, pages 166–194, 2009.

[113] S. Mödersheim and L. Viganò. Secure pseudonymous channels. *Computer Security–ESORICS 2009*, pages 337–354, 2009.

[114] D. Molnar, A. Soppera, and D. Wagner. A scalable, delegatable pseudonym protocol enabling ownership transfer of RFID tags. In *Selected Areas in Cryptography*, pages 276–290. Springer, 2006.

[115] B. Monahan. Introducing aspect-a tool for checking protocol security. Technical report, Technical Report HPL-2002-246, HP Labs, 2002.

[116] J.H. Moore. Protocol failures in cryptosystems. *Proceedings of the IEEE*, 76(5):594–602, 1988.

[117] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):999, 1978.

[118] D.M. Nessett. A critique of the burrows, abadi and needham logic. *ACM SIGOPS Operating Systems Review*, 24(2):35–38, 1990.

[119] B.C. Neuman and T. Ts'o. Kerberos: An authentication service for computer networks. *Communications Magazine, IEEE*, 32(9):33–38, 1994.

[120] C. Ng, W. Susilo, Y. Mu, and R. Safavi-Naini. RFID privacy models revisited. *Computer Security-ESORICS 2008*, pages 251–266, 2008.

[121] C.S. Ong, K. Nahrstedt, and W. Yuan. Quality of protection for mobile multimedia applications. In *Multimedia and Expo, 2003. ICME'03. Proceedings. 2003 International Conference on*, volume 2, pages II–137. IEEE, 2003.

[122] R.I. Paise and S. Vaudenay. Mutual authentication in RFID: security and privacy. In *Proceedings of the 2008 ACM symposium on Information, computer and communications security*, pages 292–299. ACM, 2008.

[123] S. Pancho. Paradigm shifts in protocol analysis. In *Proceedings of the 1999 workshop on New security paradigms*, pages 70–79. ACM, 1999.

[124] D.G. Park, C. Boyd, and E. TDawson. Classification of authentication protocols: A practical approach. *Information Security*, pages 194–208, 2000.

[125] K. Paterson and A. Yau. Cryptography in theory and practice: The case of encryption in ipsec. *Advances in Cryptology—EUROCRYPT 2006*, pages 12–29, 2006.

[126] L.C. Paulson. The inductive approach to verifying cryptographic protocols. *Journal of computer security*, 6(1-2):85–128, 1998.

[127] L.C. Paulson. Inductive analysis of the internet protocol tls. *ACM Transactions on Information and System Security (TISSEC)*, 2(3):332–351, 1999.

[128] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 245–254. ACM, 2000.

[129] G. Polya. *How to solve it: A new aspect of mathematical method*. Princeton University Press, 2008.

[130] C. Pöpper, N. Tippenhauer, B. Danev, and S. Capkun. Investigation of signal and message manipulations on the wireless channel. *Computer Security–ESORICS 2011*, pages 40–59, 2011.

[131] P. Rogaway. Formalizing human ignorance. *Progress in Cryptology-VIETCRYPT 2006*, pages 211–228, 2006.

[132] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In *Fast Software Encryption*, pages 371–388. Springer, 2004.

[133] A. W. Roscoe. Intensional specifications of security protocols. In *CSFW '96: Proceedings of the 9th IEEE workshop on Computer Security Foundations*, page 28, Washington, DC, USA, 1996. IEEE Computer Society.

[134] P.Y.A. Ryan and S.A. Schneider. An attack on a recursive authentication protocol a cautionary tale. *Information Processing Letters*, 65(1):7–10, 1998.

[135] S. Sarma, S. Weis, and D. Engels. RFID systems and security and privacy implications. *Cryptographic Hardware and Embedded Systems-CHES 2002*, pages 1–19, 2003.

[136] M. Satyanarayanan. Integrating security in a large distributed system. *ACM Transactions on Computer Systems (TOCS)*, 7(3):280, 1989.

[137] P.A. Schneck and K. Schwan. Dynamic authentication for high-performance networked applications. In *Quality of Service, 1998.(IWQoS 98) 1998 Sixth International Workshop on*, pages 127–136. IEEE, 1998.

[138] S. Schneider. Security properties and csp. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*, pages 174–187. IEEE, 1996.

[139] J.M. Seigneur, S. Farrell, C. Jensen, E. Gray, and Y. Chen. End-to-end trust starts with recognition. *Security in Pervasive Computing*, pages 251–255, 2004.

[140] S.A. Shaikh, V.J. Bush, and S.A. Schneider. Specifying authentication using signal events in CSP. *Computers & Security*, 28(5):310–324, 2009.

[141] V. Shoup. *On formal models for secure key exchange.* Citeseer, 1999.

[142] D.X. Song, S. Berezin, and A. Perrig. Athena: a novel approach to efficient automatic security protocol analysis. *Journal of Computer Security*, 9(1/2):47–74, 2001.

[143] ISO standard. *Entity authentication mechanisms; Part 1: General model, ISO/IEC 9798-1, Second Edition,.* 1991.

[144] ISO standard. *ISO/IEC 9798-3:1998, Information technology—Security techniques—Entity Authentication—Part 3: Mechanisms using digital signature techniques*. 1998.

[145] ISO standard. *SO/IEC 9798-4:1999, Information technology—Security techniques—Entity Authentication—Part 3: Mechanisms using a cryptographic check function*. 1999.

[146] ISO standard. *ISO/IEC 9798-2:2008, Information technology—Security techniques—Entity Authentication—Part 2: Mechanisms using symmetric encipherment algorithms*. 2008.

[147] S.G. Stubblebine and R.N. Wright. An authentication logic with formal semantics supporting synchronization, revocation, and recency. *Software Engineering, IEEE Transactions on*, 28(3):256–285, 2002.

[148] Y. Sun and A. Kumar. Quality-of-protection (qop): A quantitative methodology to grade security services. In *Distributed Computing Systems Workshops, 2008. ICDCS'08. 28th International Conference on*, pages 394–399. IEEE, 2008.

[149] P. Syverson and I. Cervesato. The logic of authentication protocols. *Foundations of Security Analysis and Design*, pages 63–137, 2001.

[150] P.F. Syverson and P.C. Van Oorschot. On unifying some cryptographic protocol logics. In *Research in Security and Privacy, 1994. Proceedings., 1994 IEEE Computer Society Symposium on*, pages 14–28. IEEE, 1994.

[151] P.F. Syverson and P.C. Van Oorschot. A unified cryptographic protocol logic. Technical report, DTIC Document, 1996.

[152] G. Tsudik. A family of dunces: Trivial RFID identification and authentication protocols. In *Proceedings of the 7th international conference on privacy enhancing technologies*, pages 45–61. Springer-Verlag, 2007.

[153] P. van Oorschot. Extending cryptographic logics of belief to key agreement protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 232–243. ACM, 1993.

[154] S. Vaudenay. On privacy models for RFID. *Advances in Cryptology–ASIACRYPT 2007*, pages 68–87, 2007.

[155] D. Volpano, C. Irvine, and G. Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2-3):167–187, 1996.

[156] T.Y.C. Woo and S.S. Lam. Authentication for distributed systems. *Computer*, 25(1):39–52, 1992.

[157] T.Y.C. Woo and S.S. Lam. A semantic model for authentication protocols. In *Research in Security and Privacy, 1993. Proceedings., 1993 IEEE Computer Society Symposium on*, pages 178–194. IEEE, 1993.

[158] T.Y.C. Woo and S.S. Lam. A lesson on authentication protocol design. *ACM SIGOPS Operating Systems Review*, 28(3):24–37, 1994.